'68' MICRO JOURNAL

2.00

VOLUME I ISSUE 3 • Devoted to the 6800 User • May 1979

"Small Computers Doing Big Things."

SERVING THE 6800 USERS WORLDWIDE

PHOTO CREDIT: NASA

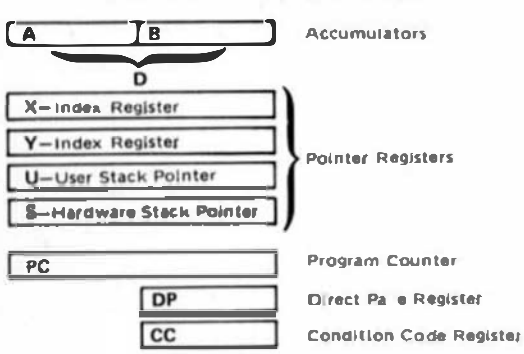
The World's Most Powerful 8-Bit Microcomputer

Featuring the World's Most Powerful 8-Bit MPU - The Motorola MC-6809

Welcome to a whole new world of microcomputing. Here at last is a microcomputer with all the speed and power that you have wished for. The MC6809 is an exciting new concept in microprocessors that fills the gap between 8- and 16-bit machines. It provides the power of 16-bit instructions with the economy of 8-bit architecture.

The MC6809 has more addressing modes than any other 8-bit processor. It has powerful 16-bit instructions, and a highly efficient internal architecture with 16-bit data paths. It is easily the most powerful, most software efficient, and the fastest 8-bit general purpose microprocessor ever.

6809 PROGRAMMING MODEL



The greatest impact of the Motorola MC6809 undoubtedly will be software related. Ten powerful addressing modes with 24 indexing sub-modes, 16-bit instructions and the consistent instruction set stimulate the use of modern programming techniques. Such as structured programming, position independent code, re-entrancy, recursion and multitasking.

A memory management system with extended addressing designed into the bus system controls up to 256K bytes of RAM memory. The dynamic memory allocation system, which is part of the multitasking DOS, allocates available memory in as small as 4K blocks.

The MC6809 system is the only 8-bit processor designed for the efficient handling of high-level languages. New addressing modes, a consistant instruction set and easy data manipulation on stacks allows the efficient execution of block-structured high-level code as generated by a compiler like PASCAL.

MP-09 Processor Card . . . . .S 195.00
68/09 Computer w/48K $1,500.00

SOUTHWEST TECHNICAL PRODUCTS CORPORATION
219 W. RHAPSODY
SAN ANTONIO, TEXAS 78216 (512) 344-0241

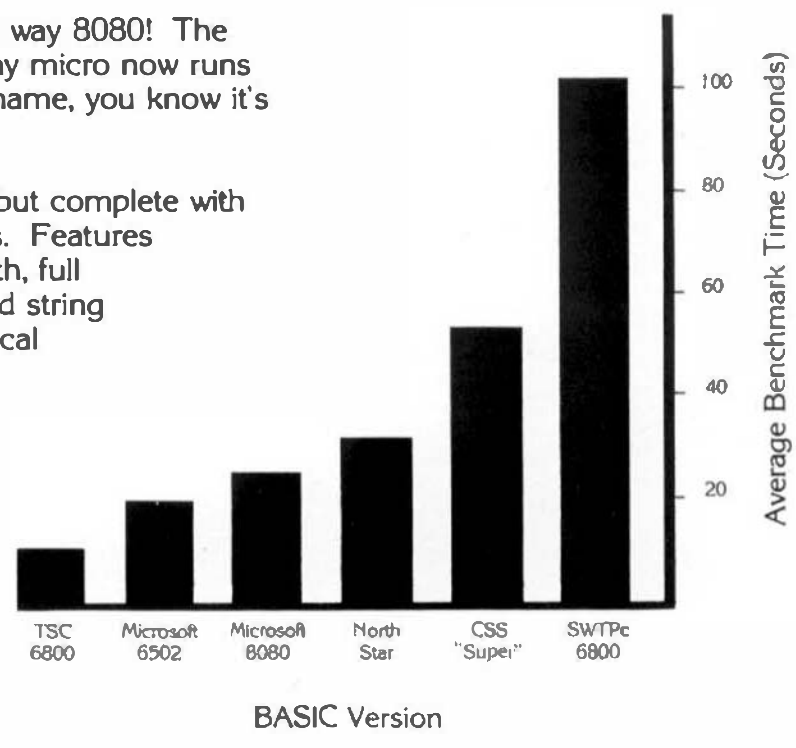
TSC BASIC for 6800

The fastest floating point BASIC for any micro.

Move over 6502! Out of the way 8080! The fastest floating point BASIC for any micro now runs on the 6800. And with the TSC name, you know it's top quality.

TSC BASIC is not only fast, but complete with over 50 commands and functions. Features include six digit floating point math, full transcendental functions, unlimited string length, if/then/else construct, logical operators,and two-dimensional arrays (including string arrays).

Available now on KCS cassette for $39.95. Requires 9K minimum, no source listing included. Soon to come is a version for the FLEX™ disk operating system.



Graph based on benchmarks listed in October 1977 issue of Kilobaud™ magazine.

Technical Systems Consultants, Inc.
Box 2574 W. Lafayette, IN 47906

Specialists in Software & Hardware for Industry & the Hobbyist

'68' MICRO JOURNAL

Portions of the text of '68' Micro Journal set using the following:
6800/2, DMAF1 and CT-82
Southwest Technical Products Corp.
219 W. Rhapsody
San Antonio, TX 78216

Editor, Word Processor and Sort/Merge
Technical Systems Consultants, Inc.
Box 2574
W. LaFayette, IN 47906

*MINIFLEX & FLEX REG.™ Technical Systems Consultants, Inc.

Selectric I/O
World Wide Electronics, Inc.
130 Northwestern Blvd.
Nashua, NH 03060

Publisher/Editor
Don Williams Sr.

Assistant Editor — Software
Mickey E. Ferguson

Assistant Editor — Hardware
Dennis Womack

Associate Editor — Southwest
Dr. Jack Bryant

Associate Editor — Midwest
Howard Berenbon

Subscriptions and Office Manager
Joyce Williams

Typography and Color Separations
Williams Company, Inc.
Chattanooga, TN 37421

CONTENTS

Send All Correspondence To:

'68' Micro Journal
3018 Hamill Rd.
PO Box 849
Hixson, Tennessee 37343

'68' Micro Journal is published 12 times a year by '68' Micro Journal, 6131 Airways Blvd., Chattanooga, TN 37421. Second Class postage paid at Chattanooga, TN. Postmaster: Send Form 3579 to '68' Micro Journal, PO Box 849, Hixson, TN 37343.

Subscription Rates U.S.A.:
(Special Charter Rate)
1 year $10.50
2 years $18.50
3 years $26.50

To increase soon!

Lifetime $125.00 (one-time payment) Twice rate shown above for Air Mail/First Class anywhere in the U.S.A.

—ITEMS SUBMITTED FOR PUBLICATION—

(Letters to the Editor for Publication) All 'letters to the Editor' should be substantiated by facts. Opinions should be indicated as such. All letters must be signed. We are interested in receiving letters that will benefit or alert our readers. Praise as well as gripes is always good subject matter. Your name may be withheld upon request. If you have had a good experience with a 6800 vendor please put it in a letter. If the experience was bad put that in a letter also. Remember, if you tell us who they are then it is only fair that your name 'not' be withheld. This means that all letters published, of a critical nature, cannot have a name withheld. We will attempt to publish 'verbatim' letters that are composed using 'good taste.' We reserve the right to define (for '68' Micro) what constitutes 'good taste.'

(Articles and items submitted for publication) Please, always include your full name, address, and telephone number. Date and number all sheets. TYPE them if you can, poorly handwritten copy is sometimes the difference between go, no-go. All items should be on 8X11 inch, white paper. Most all art work will be reproduced photographically, this includes all listings, diagrams and other non-text material. All typewritten copy should be done with a NEW RIBBON. All hand drawn art should be black on white paper. Please no hand written code items over 50 bytes. Neatly typed copy will be directly reproduced. Column width should be 3¼ inches.

(Advertising) Any Classified: Maximum 20 words. All single letters and/or numbers will be considered one (1) word. No Commercial or Business Type Classified advertising. Classified ads will be published in our standard format. Classified ads $7.50 one time run, paid in advance.

Commercial and/or Business advertisers please write or phone for current rate sheet and publication lag time.

GIMIX inc.

1337 W. 37th Place • Chicago, Illinois 60609
(312) 927-5510 TWX 910-221-4055

See GHOST Ad page 43

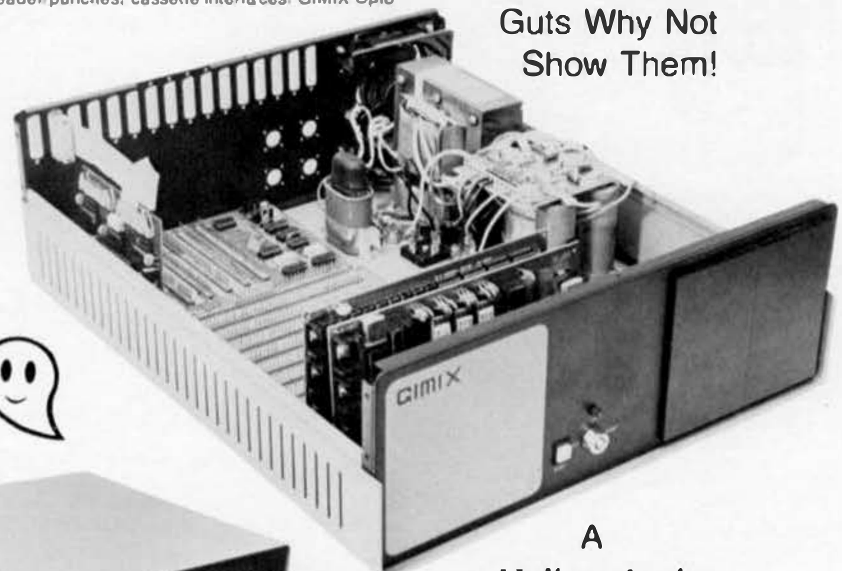
SYSTEM 68 by GIMIX

The heavyweight Champ of professional quality for the user who demands the best.

CABINET Heavy weight aluminum painted inside and out in grey and black baked enamel finish.
- Punched for 16 D type data connectors. 4 video connectors and slotted for ribbon cables
- Ventilation grooves pull fresh air over boards first
- Size — 18" wide x 21" deep x 7" high

GIMIX I/O cards provide the necessary hardware to link your computer to peripheral devices. Serial Interface cards are used with remote terminals, modems. GIMIX Relay Driver Boards, etc. Parallel Interface cards are needed for keyboards, paper tape, reader/punches, cassette interfaces. GIMIX Opto Boards, GIMIX Tone Receiver Boards, etc.

If You Have The Guts Why Not Show Them!



It's Capacious It has the capability to handle *advanced* software. It has the capacity to contain large programs.

Software Insurance Even in industrial environments, when loads such as welders or compressors start up that cause lights to dim or other computers to bomb out, the power supply of the GIMIX system keeps running glitch free. It has power for itself plus power to drive your peripherals.

Power Supply designed to power a fully loaded system plus two 5¼" disk drives — and keep running at constant voltage outputs even under adverse A.C. power input conditions. It consists of: *A 550 VA Ferroresonant constant voltage transformer* over 16 pounds of **brute force** custom designed for GIMIX to GIMIX' specs, an A.C. resonant capacitor, 3 D.C. filter capacitors, and GIMIX' unique filter assembly board that sits on top of the filter capacitors and contains individual fuses for each output, bleeder resistors, and a clamping terminal block for easy wiring connections. Almost a Quarter-farad of D.C. Filtering. *Brown-out and overvoltage insurance.* Supplies 8V at 25 Amps, + 15 Volts at 5 Amps and -15 Volts at 5 Amps from A.C. input voltages ranging from 90 to 140 Volts.

- A.C. fuse holder
- Removeable A.C. cord
- 70 CFM exhaust fan

Holds 2 5¼ disk drives (not included)
GIMIX disk regulator cards mount on drives and wire to filter assembly board (optional)

A Hallmark of Reliability!

- Optional filler plates (when no drive is used)
- 3 position keyswitch

Off/Power On, Reset Off/Both On

- Reset can be locked out

GHOSTable Features The ability of a board, or a block of memory on a board, to appear, disappear, or reappear on the bus. This allows you to have multiple memory blocks located at the same address. Using the software controllable registers on GIMIX GHOSTable boards, you can enable one block of memory at a time while disabling the other blocks. Ideal for multi-user and multi-tasking users.

Mother Board
Hardware reconfigurable to give you the utmost versatility for use of various SS 50 bus software packages. *Gold plated* pins to insure long lasting electrical contact and for protection against corrosion. Fifteen 50 pin slots plus eight DIP-switch addressable 30 pin I/O slots configurable to 4 or 8 decoded addresses. The fully buffed I/O block is addressable by DIP-switch to any 32 or 64 byte boundary and can also be disabled. UD1 and UD2 of the 50 pin bus can be strapped to UD3 and UD4 of the 30 pin bus. A fully shielded, double sided P.C. board made of .090 thickness with noise reducing ground lines on the bottom side that separate all data, address, and signal lines, and a full ground plane on the top side. A 14 posistion clamping terminal block for all power and other external connections. Eliminates soldering, crimping or forming of wires.

VIDEO BASED The ultra high speed output of GIMIX' VIDEO boards, instantly display the entire written screen. It is limited only to the entire systems software limits, whereas a terminal limits the user to the intelligence of the terminal itself. You can have multiple monitors, remote displays, and use several video boards in the same system.

Featuring GMXBUG Our video based ROM monitor (installed on the CPU board) contains all the standard utility functions found on other MIKBUG compatible monitors PLUS advanced debugging, utility and memory manipulation routines for easier software development. A video board, GMXBUG, a parallel keyboard, and a video monitor give you a complete video based system. No terminal is required.

Quality All boards assembled and tested — 100% solder masked, using only top quality components designed for lowest power consumption and coolest operation. We take no short cuts to insure that we have the highest quality possible. Complete with Connectors. All necessary instructions and documentation included.

Value — GIMIX includes as standard equipment features that are optional, if available at all on other manufacturer's systems. Our mainframe has the power supply and space for your disks; our video board is better than and eliminates a terminal. Expandibility is already there. When you add up your total system costs, you'll find that GIMIX gives you the most for your money.

SS 50 BUS BOARDS

16K Systems

Includes: Mainframe cabinet, mother board, power supply, fan, CPU, 16K static RAM, and choice of I/O card.

$1294.29

Other packages available.

16K Static RAM Boards

Gold Bus connectors — DIP switch controllable addressing, write protect and enabling of each 4K block. Tested at 2 MHz. Assembled

$298.13

Above but socketed and with software control registers Assembled

$368.16

GIMIX designs, manufacturers and tests, in-house their complete line of products. Complete systems are available to fit your needs. Please contact the factory if you have any special requirements.

See your GHOST Dealer or
ORDER DIRECTLY BY PHONE OR MAIL

GIMIX inc.

1337 W. 37th Place • Chicago, Illinois 60609
(312) 927-5510 TWX 910-221-4055

The Company that delivers.
Quality Electronic products since 1975.

THE CASE FOR SMALL DISK OPERATING SYSTEMS

By Harold Mauch, president, PERCOM DATA COMPANY

"Both the locomotive & the burro carry freight but which is the better choice for a narrow mountain path?"

I'm not against large disk operating systems for small computers. In fact, Percom has developed and markets an exceptionally powerful DOS called INDEX™. But, if you're buying a boat for water skiing, do you really need the Queen Mary? The case for the modest disk operating system for the small computer user deserves its "day in court."

Most disk operating systems for small computers have been written by programmers whose experience was developed on large — in fact, very large — computers.

Unfortunately, these selfsame programmers become so enamoured by the grandness of large computer operating systems that they drift far from the economics and the application realities of the small computer user.

"Precedence is often substituted for thinking."

THEDOS was created by the mythical programmer Wong Gotu, Senior Lead Systems Programmer in charge of Disk Operating Systems.

THEDOS is all things to all men — er, to all persons. And Mr. Gotu, having blessed the world of small computer users with a system that does everything but wipe your brow, cannot understand the frustration of S. Computer User, who, having laid out several hundred bucks for THEDOS and supporting software, now finds he must dedicate at least one drive merely to serve his magnificent operating system. Nor can Mr. Gotu (Senior Lead Systems Programmer in charge of Disk Operating Systems) comprehend the annoyance of S. Computer User when S. learns that he must devote a major part of his all-too-limited memory to "clothe" THEDOS.

After all, Wong asks, did not he provide S. Computer User with a DOS just like the big computers?

"Big Computer! I bought a micro to get away from the expense and complexity of a Big Computer!" snorts S. Computer User.

"All I want is a machine to keep my books and track my inventory. For what do I need a fancy system which can track umpteen file names? Let one disk hold the program, another hold the books. What could be simpler? I'll just write the name on the disk jacket and know exactly what is what."

So we ask: who besides the large computer user needs a complex disk operating system?

It certainly is not the small businessman. The small cost and limited storage capacity of a mini-floppy diskette as well as the need for archival backup dictate that seldom will more than one applications program or data file be stored on a given diskette. Hence, the presence of a complex DOS only diminishes the already limited disk and memory resources.

It certainly is not the cost-conscious computer hobbyist. Not when he learns he must give up a sizeable chunk of memory to THEDOS, and that he needs two or more drives because THEDOS demands the constant attention of one drive. It is only fair if he asks for a better way. After all his cassette works fine, it's just slow.

There are owners of microcomputers who have a genuine need for powerful disk operating systems, and I'll discuss these situations momentarily. First, however, I'd like to expand on the case for small operating systems.

Let's look at what we expect or want from a disk operating system:

- Permit named files
- Provide a directory
- Speed
- Save files from memory
- Load files from disk to memory
- Handle free sectors on disk

Think about it. Do you really need more than this from your DOS?

If 99% of your programming is done in BASIC you want to create files you can find and know what else is on the disk. Moreover, if your BASIC is an extended disk BASIC it already handles most of the features you want from your operating system.

If, on the other hand, most of your programming is with an Editor and Text Processor, again you want only to save and find your files.

In cases when high-level languages are used predominately, the major function of the DOS is to use up your memory! If you have less than 32K bytes of RAM that's usually more than one-fourth of it.

Furthermore, as a DOS gets larger it takes more processor time thus slowing down the actual processing of files. This can reduce processing speed by a factor of 15 or more.

You might argue that a small DOS reduces the disk to little more than a fast cassette. Is a system that loads a program more than five times faster than a complex DOS, supports Random Access data files, and uses virtually no memory just "little more than a fast cassette"? Hardly. The penalty of complexity is slower operation. If the complex DOS supports random access files it does so at the expense of disk and memory space.

Yes, I've raised an obvious question: How can one implement random access data files with a limited DOS? How? Simply insist that all blocks in a given file be consecutive and contiguous.

To illustrate random access implementation, consider a disk BASIC with the following commands:

SCTR	— returns the number of the most recently accessed disk block
RESTORE #F,N,P	— positions the specified disk file "#F" to the "Nth" block of the file beginning at element "P"

For example, to read the 9th element in the 35th block of a file called "DATA" on drive #2:

```
10 OPEN #10,"2/DATA"
20 I=SCTR
30 RESTORE #10,I+34.09
40 READ #10,A$
50 PRINT A$
60 CLOSE #10
```

SCTR returns the number of the first block of the file. Since the disk file blocks are contiguous, the desired offset from the beginning of the file is simply added to the value returned by SCTR.

Now, getting back to the question concerning who needs a complex DOS, two types of users come to mind.

One is the systems programmer. But then the systems programmer wrote THEDOS. His view that THEDOS is ideal for everyone is narrow and his arguments for a complex DOS are usually self serving.

Like the model railroader, some computer hobbyists like to emulate large computing systems. While this is certainly a justifiable pursuit for the hobbyist, it does not reflect good systems judgement when practical applications for the small computer are considered.

Before I rest my case in defense of the small DOS, think about these questions: What use could be made of memory space usurped by a complex DOS? Invert a larger matrix? Sort a bigger file? How much more data could you store on the disk if the DOS utilities were not there? How much less do you pay for a system which requires one less drive?

EDITORS REMARKS

An old joke ended 'promises, promises, promises'. How many times have you heard it repeated? Seems lately it is more apropos than ever. Of all of the complaints received and heard, slow delivery is the most referred to. Some firms are more notorious than others. Fact is; some seem to start and continue on cash in advance orders, from eager micro owners. Federal regulations to the contrary, it seems more prevalent in the microcomputer industry, than any other, I know of. I personally know of manufacturers who are running slow on deliveries, despite anything they could do. Others just don't seem to care. Already the ranks are being pruned of poor products and vendors. Yet it seems for every one that fades away, three others spring up. By the time many of us find out about a shoddy product or company, it is too late for many others, or ourselves. It is difficult to sometimes tell who are the good guys. And in a couple instances the bad guys have reverted to a good guy role. Hovever, it requires time to change a black hat to white. Better to have started out a good guy.

There is another side to the coin. Sometimes the buyer just is not reasonably patient. Note, I said 'reasonably'. We had an instance of a subscriber sending his check to us on the 14th or 15th of January (we received it on the 22d of January). Less than three weeks later we received a note, sprinkled with filth and profanity, demanding his magazine or money back. Three weeks with mail delivery time included, is not reasonable time. We promptly refunded his check but the fact remains that in his eyes, we are the bad guys. All this despite the fact that we included every subscription received in the first mailing, up to the final 24 hours. Most magazines warn that it takes six to eight weeks, at the earliest, to process a new subscription. We know how hard we all worked to get everyone included, but if others don't, then we get a black eye, deserved or not. We came out a month later than we had intended. We could have been on time but the first issue would have been printed on newsprint. We wanted to do better, so we reordered paper (correctly ordered the first time), and humped to get out within 5 working days after receiving the right paper.

This brings us to a point worth considering. If you feel that you have been overly delayed, or received poor merchandise from a 6800 dealer, let us know. Maybe we can help. This can sometimes save us all a lot of grief.

Already we have refused advertising from dealers we feel wear grey or black hats. This hurts; we need advertisers to continue to grow. More important we need readers. If we don't do our part to keep the game clean, then we don't deserve your support. I have a big stake in this publication. Our publishing house is over 66 years old. We hate striking out. It is our sincere belief that 6800 users and 6800 dealers need their own publication. However; we can only do so much, the rest is up to you.

If we all focus on a standard protocol for our particular corner of the microcomputer field, then we will succeed, where others failed. Many of the 'biggies' of the early days are gone. Some others will follow. All because they bickered and squabbled about busses and standards; until there were none.

So it all boils down to this. As users we need to share our knowledge and advancements with others less skilled (we all started on base one somtime or another). If you have a program or hardware project, let us all know. You will get your subscription extented and a lot of other folks will be grateful, not to mention it will make you an internationally known author.

A few words to hardware and software manufacturers also; please lets keep order among the 6800 products to come. Competition is one thing, suicide another. Surely the mistakes of some who have fallen, should not be repeated, in the name of competition. One of the strongest things going for 6800 user acceptance is compatability. If a fair and reasonable bus and signal protocol is agreed to by you who make the things we buy, hardware and software alike, there will be plenty for all. Screw it all up; kiss it all goodby. Price has never been the deciding factor for most, providing it is

reasonable...dependable operation and compatability have. With the 6809 and other new products coming, NOW is the time for decisions. Any comments, anyone?

CRUNCHERS CORNER

This monthly column is intended to provide a place for the exchange of ideas on microcomputer arithmetic. A systematic exposition of fixed and floating point arithmetic, hardware and software, algorithms for approximation and so on is planned. Questions and comments submitted to this column can be on any subject relevant to "number crunching," and should be addressed to:

Jack Bryant
Department of Mathematics
Texas A&M University
College Station, Texas 77843

We ask that all correspondents supply their names and addresses.

AN INTEGER ARITHMETIC PACKAGE

Earlier in this column we discussed some of the difficulties which go with using a microcomputer to solve real world problems. The central difficulty is the data width (that is, the number of bits in the data bus). The processor we are using as an example has 8 bit data. Since this only gives 256 distinct data values, we implement in software (that is, in a program) wider data types. We selected 16 bit integer arithmetic to start with; this system is simple but is also useful. Using the instructions ABC and SBC of the M6800 instruction set, we easily programmed 16 bit addition and subtraction. We also considered the conversion from ASCII sign-magnitude decimal notation to 16 bit two's complement binary, and found that this could be done efficiently with a special purpose multiply by 10 program. (Efficiency is especially critical in an interpretative program such as BASIC which must perform the conversion each time a line is executed.)

In this month's column we introduce new operations multiplication and division. We also give a program which performs conversion from 16 bit binary to ASCII sign-magnitude decimal. These more complex operations make use of the M6800 shift operations ASR, ROR, ASL and ROL. The resulting set of programs is adequate to support arithmetic operations needed for indexing (that is, for finding elements in vectors or arrays). Also, some versions of BASIC (such as the famous TINY BASIC of Tom Pittman) use exactly this data type everywhere. A unified set of programs meant to be used by other programs is sometimes called a package: this month's column contains a comprehensive 16 bit integer arithmetic package.

Before we begin to discuss details of the programs, we need to clarify a loose end from last time: it is the problem of overflow.

MODULAR ARITHMETIC

Last time we pointed out that arithmetic using two's complement or unsigned representations of binary numbers needs the same action by the internal arithmetic-logic unit in the computer. Consider once more 4 bit data: there are $2^4 = 16$ possible values, indicated in Table 1. Also shown there are other integers. For example, the integer 30 is listed with the bit pattern named 14 (as an unsigned integer) or -2 (as a two's complement integer). Looking at the table, we see that the difference between a pair of elements in the same row is a multiple of 16, and that the elements increase by 1 each time going down columns. Before continuing, it is instructive to try to answer: what should 5x6 be in this 4 bit system as an unsigned integer and as a two's complement integer?

Overflow in addition is much less frequent than in multiplication, and for that reason we have postponed this discussion until now. Let n be an integer (such as 16) and let a and b be integers. We say a is congruent to b modulo n provided $a-b$ is a multiple of n. This will certainly be the case if $a = b$, but will also be true if $a = b + n$, $a = b - n$, $a = b + 2n$ and so on. This property (being congruent modulo n) has enough of the atributes of equality so that we sometimes write "=" instead of "congruent to . . . modulo n". Taking this liberty, we see that in the four bit system 5x6 = 14 (unsigned) and 5x6 = -2 (two's complement). This is because 30, 14 and -2 are all congruent modulo 16.

Table 1.

Four Bit Binary Integers

0000	-16	0	16	32
0001	-15	1	17	33
0010	-14	2	18	34
0011	-13	3	19	35
0100	-12	4	20	36
0101	-11	5	21	37
0110	-10	6	22	38
0111	- 9	7	23	39
1000	- 8	8	24	40
1001	- 7	9	25	41
1010	- 6	10	26	42
1011	- 5	11	27	43
1100	- 4	12	28	44
1101	- 3	13	29	45
1110	- 2	14	30	46
1111	- 1	15	31	47

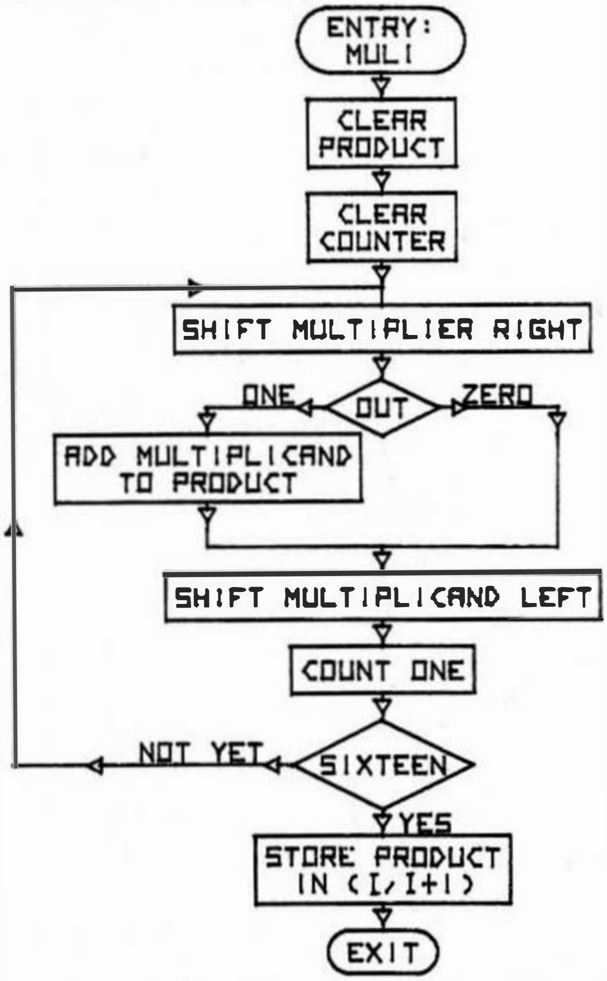
Computations with n bit integers such as addition and multiplication are carried out modulo 2^n. For $n = 16$, this is $2^{16} = 65536$. While 65536 is a lot larger than 256, it is very likely that many multiplication operations will overflow. For example, the 16 bit product 128 x 512 is equal (congruent) to 0 (modulo 65536)! Thus there are non-zero numbers which divide 0; this may take some getting used to. The important thing to remember now is that these overflows are a result of a fixed data width and are inescapable. None of the programs in the package give an alarm when this kind of overflow has occurred.

MULTIPLICATION

Consider again four bit integers. In Fig. 1 we show three four bit multiplications, exactly like multiplication of decimal numbers is carried out by hand. We show the unsigned and two's complement names for the multiplier and multiplicand, and the standard decimal name for the product. Notice that the decimal product in each case is congruent to the rightmost

four bits of the binary product modulo 16. It is this method which we develop into an algorithm for performing 16 bit integer multiplication in Flow Chart 1.

The corresponding program to implement this algorithm is named MULI in Listing 1. It replaces the contents of I by I times M, and in the process destroys M. The index register is used as a counter since both accumulators are used to build the product. The shift operations ROR, ROL and ASL are used to test bits of M and to position the contents of I correctly for adding to the partial product (if needed) at the next stage. About 650 cycles are required for one multiplication.

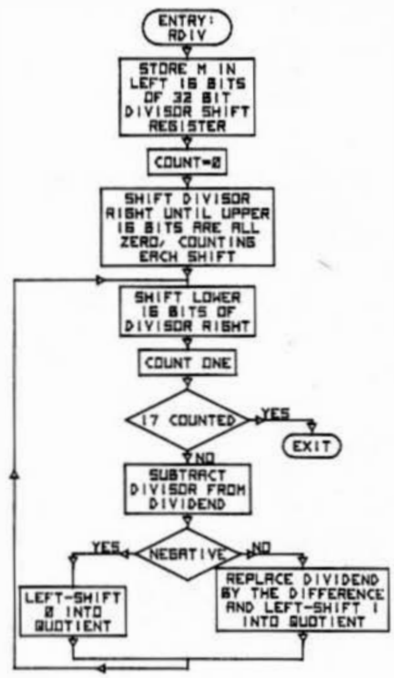


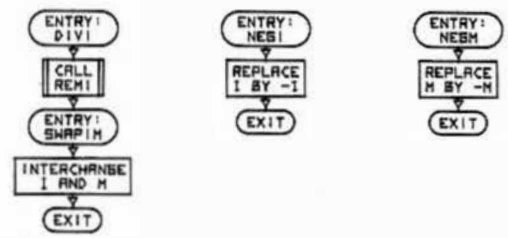

```
 3    3     0011          8   -8     1000
 4    4     0100          8   -8     1000
--   --     ----         --   --     ----
12   12      0000        64   64      0000
            0000                     0000
           0011                     0000
          0000                     1000
          -------                  -------
          0001100                  1000000


14   -2     1110
 7    7     0111
--   --     ----
98  -14      1110
            1110
           1110
          0000
```

Fig. 1. Three Multiplications of Four Bit Unsigned Binary Integers. Note that the right-most four bits of the answer is correct modulo 16.

	Example a: 7/2			Example b: 5/3		
	divisor	dividend		divisor	dividend	
	00100000	00000111		00110000	00000101	
1. Shift divisor	00010000			00011000		
Subtract and test	negative		0	negative		0
2. Shift divisor	00001000			00001100		
Subtract and test	negative		00	negative		00
3. Shift divisor	00000100			00000110		
Subtract and test	positive:	00000011	001	negative		000
4. Shift divisor	00000010			00000011		
Subtract and test	positive:	00000001	0011	positive:	00000010	0001
		remainder 1	quotient 3		remainder 2	quotient 1

Fig. 2. Two Four Bit Divisions With Remainder





DIVISION

Although there is a sense in which division is the inverse operation to multiplication, it is easy to see that the problems involved are quite different. In multiplication, the principle problem is the product often overflows. This is solved using modular arithmetic. In division, the problem is the division cannot be performed exactly. That is, the divisor does not "go into" the dividend. This problem is handled using a remainder. Let a and b be integers with a not equal to zero. In division, we wish to

write $b = q * a + r$ where q is the quotient b/a and r is the remainder. In sign-magnitude representation, the sign of the quotient is negative when a and b have opposite signs, the sign of r is equal to the sign of b, and the magnitude $|r|$ of r satisfies $0 \le |r| < |a|$. These conditions uniquely determine q and r. In binary, the division algorithm given here requires a and b to be non-negative and builds the quotient one bit at a time, reducing the dividend by the shifted divisor each time a 'one' bit is shifted into the quotient.

In Fig. 2, we trace the steps involved on two four-bit problems: 7/2 and 5/3. Initially, we place the divisor in the left four bits of an 8 bit register A. The dividend is placed in the right four bits of an 8 bit register B. The first step is a right shift of the divisor register A. Then A is subtracted from B; if the result is negative, 0 is shifted into the quotient; otherwise 1 is shifted into the quotient and B is replaced by the difference. Then A is shifted right and the process is repeated. This process is shown in more detail in Flow Chart 2.

This algorithm does not work correctly for negative two's complement numbers. Accordingly, the dividend and divisor are made positive and the sign of quotient and remainder calculated. The program REMI in Listing 1 ends with the quotient I/M in M and the remainder in I. Program DIVI ends with the quotient in I and remainder in M (and in M + 2).

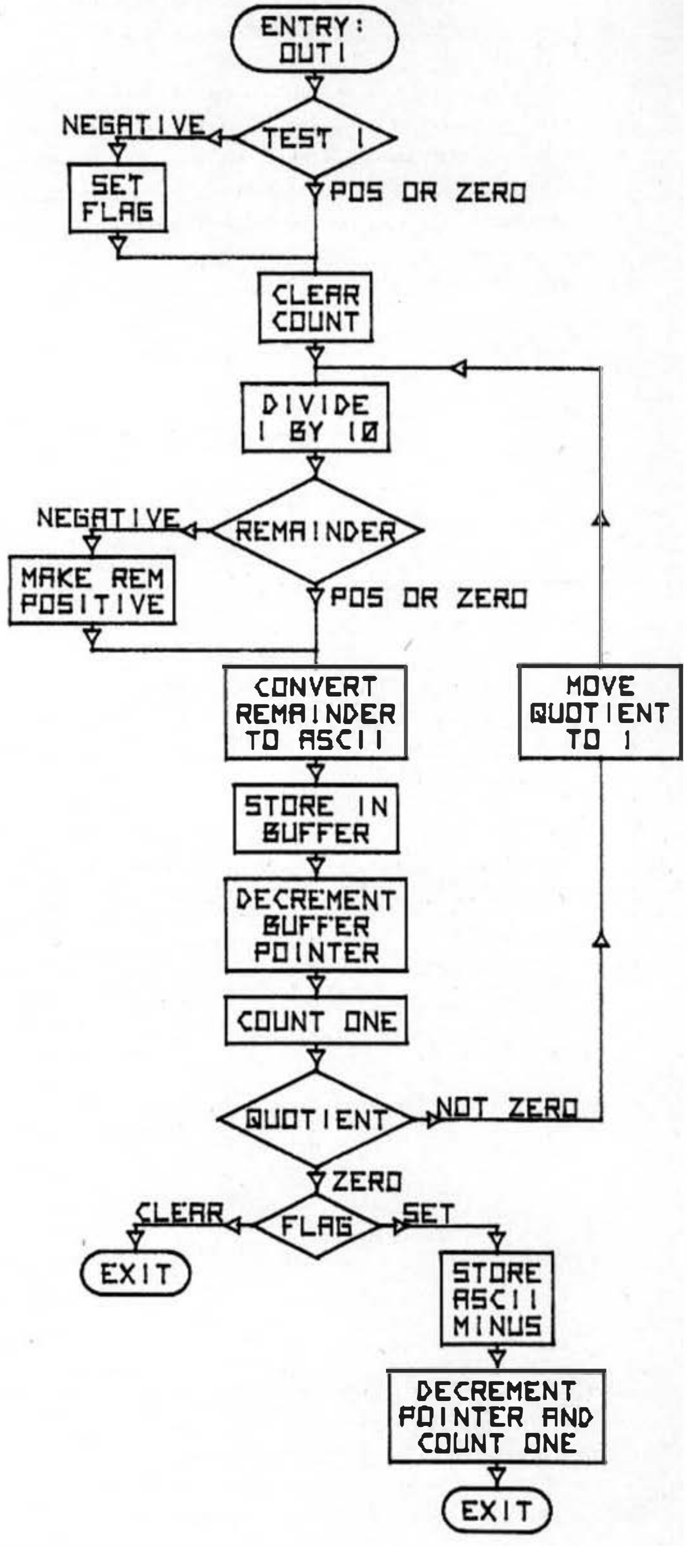
A programming note on the implementation of Flow Chart 2: For 16 bit division with remainder, the extended divisor and dividend register contain 32 bits. However, any time the left 16 bits of the divisor register is non-zero, the subtraction would result in a negative difference. Accordingly, the subtraction need not actually be carried out until the divisor has been shifted far enough right. This also saves left-shifting the quotient (since zero would be shifted in). Also, once the left-most 16 bits are zero, they need not actually be subtracted. The slight increase in program complexity results in a division program more than twice as fast. About 900 cycles are required for a typical division.

BINARY TO ASCII

The final problem we discuss this month is the conversion from 16 bit two's complement numbers to ASCII sign-magnitude decimal. Recall how base conversions go for positive numbers: to convert a number n to decimal, we may write $n = 10 * m + r$, where $0 \le r < 10$. The digit r is the least significant digit in the decimal representation of n. Repeating this process with m replacing n gives the next least significant digit and so on until the quotient m becomes zero. The digits (in ASCII) are easily generated from their 8 bit binary values by adding $30. This process is shown in Flow Chart 3, and programmed in program OUTI of Listing 1. One minor problem is that the digits are generated in reverse oder (i.e., the least significant digit is generated first). This is handled by storing the ASCII number in a biffer to be read out later. Since the result of a conversion is usually printed or displayed on a relatively slow I/O device, the execution time for OUTI is unimportant.

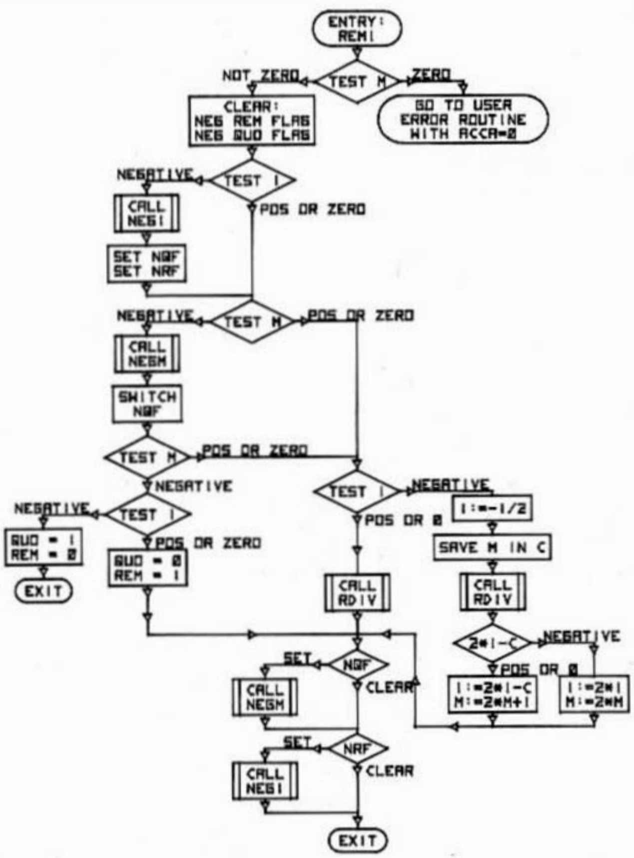
THE INTEGER ARITHMETIC PACKAGE

The program in Listing 1 is intended to illustrate each of the principle arithmetic operations on 16 bit integers. There are three main parts: the test program and its main subroutine BANG which calls the appropriate operation, the input/output (I/O) subroutines INPUT and OUTPUT which move strings from the console to memory and back, and the arithmetic operation programs themselves. In Table 2, each arithmetic subroutine in the package proper is briefly described. The entire package requires under 400 bytes. About



175 additional bytes are needed to provide the test program and I/O; these test and I/O routines are only meant to be examples of what can be done (and, of course, serve to test the various routines).

Several remarks on the philosophy of these programs are in order. As mentioned in last month's column, certain underlying programming conventions have been followed. Thus the current value of index register and B accumulator is not changed by any of these programs (unless, as is the case for READI and OUTI, the change is expected and documented). Each program tolerates interrupts (providing the interrupting program does not modify the thirteen page zero locations beginning with $F3). Also, each program will execute in read only memory (although they would obviously be assembled at a different address in most systems). Finally, each program should be as accurate as it is



possible to be. (For integer arithmetic, this means each program must be perfect.) Given these conventions, the principle goal was to minimize execution time (first) and to minimize memory requirements (second).

A good example of the execution time vs. memory problem is the integer division program in Listing 1. The algorithm chosen was the fastest known for positive operands; however, it fails for negative numbers, and, since -32768 is its own two's complement negative, something must be done to accomodate this exceptional condition. (The accuracy convention: each program must be as accurate as it can be.) There are three exceptional cases: the negative division overflow, the negative dividend overflows, or both. Program REMI is complicated because the program RDIV to perform division is fast and simple. All this logic takes about 80 bytes, although these instructions are rarely executed.

Next month we give examples of usage of the programs in the integer arithmetic package. Also, alternate methods of coding the programs are discussed.

```
INTEGER ARITHMETIC PACKAGE V 2.0            '68 MICRO JOURNAL     PAGE 1

                  *       MIKBUG EQUATES
                  *
E07E              PDATA1 EQU  $E07E   BLOCK OUTPUT
E1AC              INEEE  EQU  $E1AC   INPUT ONE CHARACTER
E1D1              OUTEEE EQU  $E1D1   CHARACTER OUTPUT ROUTINE
                  *
                  *       PAGE ZERO CONSTANTS
                  *
00F1                     ORG  $F1
00F1              ATEMP  RMB  1          TEMPORARY ACCUMULATOR STORE
00F2              INBUF  RMB  2          INPUT BUFFER BEGINNING
00F4              NEGFLG RMB  1          NEGATIVE INPUT FLAG
00F5              M      RMB  6          M SOFTWARE ACCUMULATOR
00FB              I      RMB  2          I SOFTWARE ACCUMULATOR
00FD              XTEMP  RMB  2          TEMPORARY STORAGE FOR INDEX REGISTER
00FF              SGNREM RMB  1          INTEGER DIVISION LOCAL
                  *
                  *       INDEXED REGISTER ADDRESSED STORAGE ON PAGE 1
                  *
0200                     ORG  $0200   TEST ROUTINE STARTS HERE
                  *
                  *       TEST PROGRAM FOR INTEGER ARITHMETIC PACKAGE
                  *             VERSION 2.0
                  *
0200 8D 3A               BSR  CRLF    START WITH A NEW LINE.
0202 8D 0E        TEST   BSR  INPUT    GET NUMBER AND OPERATOR STRING
0204 DE F2               LDX  INBUF
0206 8D 3F               BSR  BANG     PERFORM OPERATIONS
0208 CE 01 FF            LDX  #$1FF
020B BD 04 0E            JSR  OUTI
020E 8D 23               BSR  OUTP T  PRINT THE RESULT.
0210 20 F0               BRA  TEST     DO IT AGAIN
                  *
                  *       INPUT SUBROUTINE
                  *
                  *
                  *             SUBROUTINE TO TAKE A STRING FROM THE CONSOLE
                  *             AND STORE IN INPUT BUFFER.
                  *
0212 DF FD        INPUT  STX  XTEMP   SAVE INDEX REGISTER
0214 37                  PSH B        SAVE ACCB
0215 5F                  CLR B        SET COUNTER
0216 CE 01 00            LDX  #$100   INPUT BUFFER STARTS AT $100
0219 DF F2               STX  INBUF
021B BD E1 AC     INEXT  JSR  INEEE   GET A CHARACTER
021E A7 00        ISTASH STA A 0,X    STORE IT
0220 08                  INX
0221 5C                  INC B        BUMP COUNTER
0222 29 0A               BVS  IFULL   BUFFER SIZE = 127
0224 81 0D               CMP A #$D    CR?
INTEGER ARITHMETIC PACKAGE V 2.0            '68 MICRO JOURNAL     PAGE 2

0226 26 F3               BNE  INEXT   NOPE
0228 8D 12               BSR  CRLF    NEW LINE.
022A DE FD               LDX  XTEMP   RESTORE INDEX REGISTER
022C 33                  PUL B        RESTORE ACCB
022D 39                  RTS
022E 09           IFULL  DEX          BUFFER FULL; GENERATE CR.
022F 86 0D               LDA A #$D
0231 20 EB               BRA  ISTASH
                  *
                  *       OUTPUT SUBROUTINE
                  *
                  *             THIS SUBROUTINE PRINTS A STRING
                  *             POINTED TO BY THE INDEX REGISTER
                  *             PLUS 1 WITH ACCB CONTAINING THE
                  *             COUNT OF THE NUMBER OF CHARACTERS
                  *             PRINTED.  ACCB IS ZERO ON EXIT.
                  *             AND THE INDEX REGISTER IS LEFT
                  *             POINTING TO THE LAST PRINTED.
                  *
0233 08           OUTPUT INX          POINT TO THE NEXT ONE.
0234 A6 00               LDA A 0,X
0236 BD E1 D1            JSR  OUTEEE  PRINT IT
0239 5A                  DEC B        COUNT
023A 26 F7               BNE  OUTPUT
023C DF FD        CRLF   STX  XTEMP
023E CE 04 40            LDX  #CRLFD
0241 BD E0 7E            JSR  PDATA1
0244 DE FD               LDX  XTEMP
0246 39                  RTS
                  *
                  *       SUBROUTINE BANG
                  *
                  *             THIS SUBROUTINE TAKES THE STRING IN THE INPUT
                  *             BUFFER AND LOCATES THE OPERATORS AND CONSTANTS
                  *             AND CALLS THE APPROPRIATE ROUTINE TO APPLY THE
                  *             OPERATION.  THEY ARE APPLIED LEFT-TO-RIGHT
                  *             AS ENCOUNTERED.
                  *
                  *             OPERATIONS SO FAR:
                  *
                  *      +      ADDITION:       I := I + M
                  *      -      SUBTRACTION:    I := I - M
                  *      *      MULTIPLICATION:  I := I * M
                  *      /      DIVISION:       I := I / M
                  *      R      REMAINDER:      I := REMAINDER IN DIVISION I/M
                  *      A      ABSOLUTE VALUE:  I := ABS(I)
                  *      N      UNIARY NEGATION: I := -I
                  *
0247 37           BANG   PSH B
0248 8D 57               BSR  READI   CONVERT ONE NUMBER
```

Table 2. Arithmetic and Conversion Subroutines

Name	Entry Point	Purpose	Length (bytes)	Time (cycles)	Remarks
Binary Operations					
ADDI	$02E4	I:=I+M	13	25	
SUBI	$02F1	I:=I-M	13	25	
MULI	$02FE	I:=I*M	37	608 to 704	M becomes undefined
DIVI	$0340	I:=I/M M:=REMAINDER	19	about 900	Calls REMI Entry point SWAPIM
REMI	$0353	I:=REMAINDER M:=I/M	187	about 850	Calls NEGI, NEGM Length includes RDIV
Interchange Operation					
SWAPIM	$0342	Interchange I and M	17	41	
Unary Operations					
ABSI	$032F	I:=|I|	5	12 to 27	Calls NEGIEN
NEGI	$0334	I:=-I	12	24 to 26	Entry point NEGIEN
Conversion Operations					
READI	$02A1	M:=binary	67	70x(1+# of digits)	Entry: index register points to start of string. Exit: index register points to non digit, no. in M. I is not changed.
OUTI	$040E	Output value of I in decimal	57	1000x(# of digits)	Calls REMI Entry: index register points to end of at least 6 byte buffer. Exit: index register plus one points to start of ASCII string. ACCB contains count of no. of bytes (including leading minus sign for a negative number).

```
INTEGER ARITHMETIC PACKAGE V 2.0              '68 MICRO JOURNAL       PAGE 3

024A 96 F5              LDA A M      SAVE THE
 24C 97 FB              STA A I      FIRST ONE
024E 96 F6              LDA A M+1    IN I
0250 97 FC              STA A I+1    TOO.
0252 E6 00     BANGN    LDA B 0,X    LOAD THE OPERATOR
0254 08                 INX          POINT TO NEXT OPERAND
0255 C1 20              CMP B #$20
0257 2D 46              BLT   OVER   SEE IF A CONTROL CHARACTER
0259 8D 46              BSR   READI  GET THE NEXT OPERAND
025B C1 2B              CMP B #$2B
025D 26 05              BNE   SUBO
025F BD 02 E4           JSR   ADDI   ADD
0262 20 EE              BRA   BANGN
0264 C1 2D     SUBO     CMP B #$2D
0266 26 05              BNE   MULO
0268 BD 02 F1           JSR   SUBI   SUBTRACT
026B 20 E5              BRA   BANGN  DO IT AGAIN
026D C1 2A     MULO     CMP B #$2A
026F 26 05              BNE   DIVO
0271 BD 02 FE           JSR   MULI   MULTIPLY
0274 20 DC              BRA   BANGN
0276 C1 2F     DIVO     CMP B #$2F
0278 26 05              BNE   REMO
027A BD 03 40           JSR   DIVI   DIVIDE
027D 20 D3              BRA   BANGN
027F C1 52     REMO     CMP B #$52
0281 26 05              BNE   ABSO
0283 BD 03 53           JSR   REMI   REMAINDER
0286 20 CA              BRA   BANGN
0288 C1 41     ABSO     CMP B #$41
028A 26 05              BNE   NEGO
028C BD 03 2F           JSR   ABSI   ABSOLUTE VALUE.
028F 20 C1              BRA   BANGN
0291 C1 4E     NEGO     CMP B #$4E
0293 26 05              BNE   ERROR
0295 BD 03 34           JSR   NEGI   NEGATIVE OF I
0298 20 B8              BRA   BANGN
029A 86 3F     ERROR    LDA A #$3F   LOAD AN ERROR FLAG.
029C BD E1 D1           JSR   OUTEEE PRINT IT
029F 33        OVER     PUL B        RESTORE A CB
02A0 39                 RTS
               *
               *        READ SUBROUTINE
               *
               *     ENTER WITH INDEX REGISTER POINTING TO STRING
               *     TO BE CONVERTED TO 16 BIT BINARY.
               *
               *     EXIT WITH INDEX REGISTER POINTING TO NEXT NON
               *     (0-9) OR LEADING MINUS SIG  AND WITH NUMBER IN M.
               *

INTEGER ARITHMETIC PACKAGE V 2.0              '68 MICRO JOURNAL       PAGE 4

02A1 37        READI    PSH B        SAVE A CB
02A2 5F                 CLR B        CLEAR ACCUMULATOR
02A3 37                 PSH B
02A4 D7 F4              STA B NEGFLG CLEAR NEGATIVE FLAG
02A6 A6 00              LDA A 0,X    GET DIGIT/SIGN
02A8 81 2D              CMP A #$2D   LEADING MINUS?
02AA 26 05              BNE   GOON   NO
02AC 97 F4              STA A NEGFLG YES
02AE 08        NEXT     INX          PREPARE TO GET NEXT ONE.
02AF A6 00              LDA A 0,X    DO IT
02B1 81 30     GOON     CMP A #$30   >= ZERO?
02B3 2D 1E              BLT   NOTONE
02B5 81 39              CMP A #$39   <= NINE?
02B7 2E 1A              BGT   NOTONE
02B9 80 30              SUB A #$30   GOT ONE, CONVERT TO BINARY
02BB 97 F1              STA A ATEMP  SAVE FOR LATER
02BD 32                 PUL A        RESTORE MSB
02BE 58                 ASL B        *
0 BF 49                 ROL A        *
02C0 97 F5              STA A M      *
02 2 D7 F6              STA B M+1    *
02C4 58                 ASL B        *     MULTIPLY BY 10
02 5 49                 ROL A        *
02C6 58                 ASL B        *
02C7 49                 ROL A        *
02C8 DB F6              ADD B M+1    *
02CA 99 F5              ADC A M      *
02CC DB F1              ADD B ATEMP  ADD IN DIGIT
02CE 89 00              ADC A #0     AND CARRY OUT.
  D0 36                 PSH A        FREE ACCA
02D1 20 DB              BRA   NEXT   DO IT AGAIN
02D3 32        NOTONE   PUL A        HERE WE DIDN'T FIND A DIGIT.
02D4 7D 00 F4           TST   NEGFLG NEGATIVE?
02D7 27 05              BEQ   SKIP
02D9 43                 COM A
02DA 50                 NEG B        MAKE IT NEGATIVE.
02DB 26 01              BNE   SKIP
02DD 4C                 INC A        CARRY OUT
02DE 97 F5     SKIP     STA A M      STASH THE NUMBER
02E0 D7 F6              STA B M+1    WE BUILT.
02E2 33                 PUL B        RESTORE ACCB
 0 E3 39                RTS
               *
               *        SUBROUTINE ADDI
               *
               *            ADDS M TO I AND PLACES THE RESULT IN I.
               *
02E4 96 FC     ADDI     LDA A I+1
02E6 9B F6              ADD A M+1
02E8 97 FC              STA A I+1

INTEGER ARITHMETIC PACKAGE V 2.0              '68 MICRO JOURNAL       PAGE 5

02EA 96 FB              LDA A I
02EC 99 F5              ADC A M
02EE 97 FB              STA A I
02F0 39                 RTS
               *
               *        SUBROUTINE SUBI
               *
               *            SUBTRACTS M FROM I, LEAVING DIFFERENCE IN I.
               *
02F1 96 FC     SUBI     LDA A I+1
02F3 90 F6              SUB A M+1
02F5 97 FC              STA A I+1
02F7 96 FB              LDA A I
02F9 92 F5              SBC A M
02FB 97 FB              STA A I
02FD 39                 RTS
               *
               *        SUBROUTINE MULI
               *
               *            MULTIPLIES I BY M AND PLACES THE RESULT IN I.
               *
02FE DF FD     MULI     STX   XTEMP
0300 37                 PSH B          SAVE X AND B
0301 CE 00 10           LDX   #16      SET COUNTER
0304 4F                 CLR A          CLEAR PRODUCT MSB
0305 5F                 CLR B          AND LSB
0306 76 00 F5  M GN     ROR   M
0309 76 00 F6           ROR   M+1      TEST LEAST SIGNIFICANT BIT
030C 24 04              BCC   MSKIP    OF MULTIPLIER.
030E DB FC              ADD B I+1      ADD IN MULTIPLICAND
0310 99 FB              ADC A I        IF SET
0312 78 00 FC  MSKIP    ASL   I+1      SHIFT MULTIPLICAND
0315 79 00 FB           ROL   I        FOR NEXT TIME
0318 09                 DEX            COUNT ONE
0319 26 EB              BNE   MAGN     REPEAT IF NOT FINISHED
031B D7 FC              STA B I+1      STORE PRODUCT
031D 97 FB              STA A I
031F 33        RSTBX    PUL B
0320 DE FD              LDX   XTEMP    RESTORE X AND B
0322 3                  RTS
               *
               *        SUBROUTINE NEGM
               *
               *            RE LACES M BY -M.
               *
0323 96 F5     NEGM     LDA A M
0325 43        NEGMEN   COM A          ENTRY FOR WHEN M ALREADY LOADED.
0326 70 00 F6           NEG   M+1
0329 26 01              BNE   NEGMI
032B 4C                 INC A

INTEGER ARITHMETIC PACKAGE V 2.0              '68 MICRO JOURNAL       PAGE 6

032C 97 F5     NEGMI    STA A M
032E 39                 RTS
               *
               *        SUBROUTINE ABSI
               *
               *            REPLACES I BY ABSOLUTE VALUE OF I.
               *
032F 96 FB     A SI     LDA A I
0331 2D 03              BLT   NEGIEN   IF NEGATIVE MAKE IT POSITIVE.
0333 39                 RTS
               *
               *        SUBROUTINE NEGI
               *
               *            REPLACES I BY -I
               *
0334 9  FB     NEGI     LDA A I
0336 43        NEGIEN   COM A          ENTRY FOR WHEN A ALREADY LOADED.
0337 70 00 FC           NEG   I+1
033A 26 01              BNE   NEGII
033C 4C                 INC A
033D 97 FB     NEGII    STA A I
033F 39                 RTS
               *
               *        SUBROUTINE DIVI
               *
               *            REPLACES I BY I/M.  REMAINDER IS IN M.
               *
0340 8D 11     DIVI     BSR   REMI     PERFORM DIVISION
0342 DF FD     SWAPIM   STX   XTEMP    UTILITY TO SWAP I AND M.
0344 DE FB              LDX   I
0346 DF F7              STX   M+2
0348 DE F5              LDX   M
034A DF FB              STX   I
034C DE F7              LDX   M+2
034E DF F5              STX   M
0350 DE FD              LDX   XTEMP    RESTORE INDEX REGISTER.
0352 39                 RTS
               *
               *        SUBROUTINE REMI
               *
               *            REPLACES I BY THE REMAINDER FOLLOWING A DIVISI
               *            I/M.  M CONTAIN  THE QUOTIENT.
               *
0353 96 F6     REMI     LDA A M+1
0355 9A F5              ORA A M        CHECK DIVISION BY ZERO
0357 26 03              BNE   DNOERR
0359 7E 04 4B           JMP   USRERR   INT DIV BY ZERO CODE = 0
035C 37        DNOERR   PSH B          SAVE ACCB
035D DF FD              STX   XTEMP    AND INDEX
035F 5F                 CLR B           LEAR NEGATIVE QUOTIENT FLAG.

INTEGER ARITHMETIC PA KAGE V 2.0              '68 MICRO JOURNAL       PAGE 7

0360 D7 FF              STA B SGNREM   CLEAR NEGATIVE REMAINDER FLAG.
0362 9  FB              LDA A I        CHECK NUMERATOR SIGN.
0364 2C 05              BGE   DPOSI
0366 53                 COM B          SET NOF
0367 D7 FF              STA B SGNREM   SET NRF
0369 8D CB              BSR   NEGIEN   REPLACE I BY -I
036B 96 F5     DPOSI    LDA A M        CHECK DIVISOR.
036D 2C 05              BGE   DPOS3
036F 53                 COM B          TOGGLE NOF
```

```
0370 8D B3              BSR    NEGMEN   MAKE DIVISOR NEGATIVE.
0372 2D 1F              BLT    DIVINE   DON'T DIVIDE IF OVERFLOW.
0374 D7 F4     DPOS3    STA B  NEGFLG   NOW DIVIDE!
0376 7D 00 FB           TST    I        CHECK DIVIDEND OVERFLOW.
0379 2D 2A              BLT    DOVF
037B 8D 51              BSR    PDIV     OK TO DIVIDE.
               *        FIX SIGNS NOW
037D 7D 00 F4  FIXSGN   TST    NEGFLG
0380 27 02              BEQ    DPOS5
0382 8D 9F              BSR    NEGM     MAKE QUO POS.
0384 7D 00 FF  DPOS5    TST    SGNREM
0387 27 02              BEQ    DPOS6
0389 8D A9              BSR    NEGI     MAKE REMAINDER NEGATIVE.
038B 7E 03 1F  DPOS6    JMP    RSTBX
038E 78 00 F6  DNEG     ASL    M+1
0391 20 6E              BRA    DSHFT
0393 7F 00 F5  DIVINE   CLR    M        NOTE M+1 = 0
0396 7D 00 FB           TST    I
0399 2C E9              BGE    DPOS5    OK. GO FIX REMAINDER SIGN.
039B 7C 00 F6           INC    M+1      TWO OVERFLOWS.
039E 7F 00 FB           CLR    I        ZERO REMAINDER.
03A1 20 E8              BRA    DPOS6    1 QUOTIENT.
03A3 86 40     DOVF     LDA A  #$40     BINARY 0100 0000
03A5 97 FB              STA A  I        MAKE I = -(-32768/2)
03A7 DE F5              LDX    M        SAVE DIVISOR FOR LATER.
03A9 DF F9              STX    M+4
03AB 8D 21              BSR    PDIV     NOW DIVIDE.
03AD 78 00 F6           ASL    M+1      MULTIPLY QUO BY 2
03B0 79 00 F5           ROL    M
03B3 78 00 FC           ASL    I+1      MULTIPLY REM BY 2
03B6 79 00 FB           ROL    I
03B9 96 FB              LDA A  I
03BB D6 FC              LDA B  I+1
03BD D0 FA              SUB B  M+5
03BF 92 F9              SBC A  M+4      SEE IF DIVISION WAS CORRECT.
03C1 2D BA              BLT    FIXSGN
03C3 D7 FC              STA B  I+1      NO. FIX REMAINDER.
03C5 97 FB              STA A  I
03C7 DE F5              LDX    M        AND FIX QUOTIENT.
03C9 08                 INX
03CA DF F5              STX    M
INTEGER ARITHMETIC PACKAGE V 2.0          '68 MICRO JOURNAL     PAGE 8

03CC 20 AF              BRA    FIXSGN   GO CORRECT SIGNS.
               *
               *                        MAIN LOOP(S) TO PERFORM ACTUAL DIVISION
               *                        WHEN EVERYTHING IS IN ORDER.
03CE 5F        PDIV     CLR B
03CF D7 F7              STA B  M+2      CLEAR DIVISOR SHIFT AREA.
03D1 D7 F8              STA B  M+3
03D3 CE 00 10           LDX    #16      LOOP COUNT
03D6 74 00 F5  DLOOP    LSR    M
03D9 76 00 F6           ROR    M+1      SHIFT DIVISOR RIGHT.
03DC 76 00 F7           ROR    M+2
03DF 76 00 F8           ROR    M+3
03E2 09                 DEX             COUNT ONE.
03E3 96 F5              LDA A  M        CHECK IF UPPER 16 BITS ZERO.
03E5 9A F6              ORA A  M+1
03E7 26 ED              BNE    DLOOP
03E9 74 00 F7           LSR    M+2      BEGIN WITH A SHIFT
03EC 76 00 F8           ROR    M+3
03EF 96 FC     DLOOPM   LDA A  I+1      MAIN LOOP
03F1 90 F8              SUB A  M+3
03F3 D6 FB              LDA B  I
03F5 D2 F7              SBC B  M+2
03F7 2D 95              BLT    DNEG     TEST NEGATIVE
03F9 D7 FB              STA B  I        NO, POSITIVE.
03FB 97 FC              STA A  I+1      FIX DIVIDEND
03FD 0D                 SEC
03FE 79 00 F6           ROL    M+1      AND QUOTIENT.
0401 79 00 F5  DSHFT    ROL    M
0404 77 00 F7           ASR    M+2      SHIFT DIVISOR
0407 76 00 F8           ROR    M+3
040A 09                 DEX             COUNT
040B 26 E2              BNE    DLOOPM
040D 39                 RTS
               *
               *        SUBROUTINE OUTI
               *
               *                SUBROUTINE TO CONVERT FROM 16 BIT
               *                TWO'S COMPLEMENT BINARY TO ASCII
               *                SIGN-MAGNITUDE DECIMAL REPRESENTATION.
               *
               *                ON ENTRY:  INDEX REGISTER POINTS
               *                        TO END OF AT LEAST 6 BYTE BUFFER
               *                        AREA TO RECIEVE THE ASCII STRING.
               *
               *                ON EXIT:   INDEX REGISTER PLUS ONE
               *                        POINTS TO THE START OF THE STRING.
               *                ACCB CONTAINS THE COUNT OF THE NUMBER
               *                        OF BYTES IN THE OUTPUT STRING.
               *
040E 5F        OUTI     CLR B           CLEAR NEGATIVE FLAG.

INTEGER ARITHMETIC PACKAGE V 2.0          '68 MICRO JOURNAL     PAGE 9

040F 96 FB              LDA A  I        GET SIGN
0411 2C 01              BGE    OPOS1
0413 53                 COM B           SET FLAG.
0414 D7 F1     OPOS1    STA B  ATEMP    SAVE FLAG
0416 5F                 CLR B           CLEAR COUNTER.
0417 86 0A     OAGN     LDA A  #10      LOAD UP 10
0419 97 F6              STA A  M+1
041B 7F 00 F5           CLR    M        STORE 10
041E BD 03 53           JSR    REMI     DIVIDE.
0421 96 FC              LDA A  I+1      LOAD REMAINDER.
0423 2C 01              BGE    OOK
0425 40                 NEG A           HERE WHEN CONVERTING A NEGATIVE NUMBER.
0426 8B 30     OOK      ADD A  #$30     CONVERT TO ASCII.
0428 A7 00              STA A  0,X      STORE IN OUTPUT STRING.
042A 5C                 INC B           COUNT ONE.
042B 09                 DEX
042C 96 F5              LDA A  M        SEE IF OUTPUT IS ZERO YET.
042E 9A F6              ORA A  M+1
0430 26 0C              BNE    ONOTO
0432 7D 00 F1           TST    ATEMP
0435 27 06              BEQ    OPLUS
0437 86 2D              LDA A  #$2D     MINUS SIGN.
0439 A7 00              STA A  0,X
043B 09                 DEX
043C 5C                 INC B           COUNT.
043D 39        OPLUS    RTS
               *
043E 96 F5     ONOTO    LDA A  M
0440 97 FB              STA A  I        MOVE QUOTIENT TO I.
0442 96 F6              LDA A  M+1
0444 97 FC              STA A  I+1
0446 20 CF              BRA    OAGN
               *                        END OF CODE.
0448 0D        CRLFD    FCB    $D,$A,4  DATA FOR CR/LF
0449 0A 04
044B 39        USERR    RTS             USER SUPPLIED ERROR ROUTINE.

                        END
        NO ERROR(S) DETECTED
```

SYMBOL TABLE:

ABSI	032F	ABSO	0288	ADDI	02E4	ATEMP	00F1
BANG	0247	BANGM	0252	CRLF	023C	CRLFD	0448
DIVI	0340	DIVINE	0393	DIVO	0276	DLOOP	03D6
DLOOPM	03EF	DNEG	038E	DMOERR	035C	DOVF	03A3
DPOS1	036B	DPOS3	0374	DPOS5	0384	DPOS6	038B
DSHFT	0401	ERROR	029A	FIXSGN	037D	GOON	02B1
I	00FB	IFULL	022E	INBUF	00F2	INEEE	E1AC
INEXT	021B	INPUT	0212	ISTASH	021E	M	00F5
MAGN	0306	MSKIP	0312	MULI	02FE	MULO	026D
NEGFLG	00F4	NEGI	0334	NEGI1	033D	NEGIAN	0336
NEGM	0323	NEGMI	032C	NEGMEM	0325	NEGO	0231
NEXT	02AE	NOTDJE	02D3	OAGN	0417	ONOTO	043E
OOK	0426	OPLUS	043D	OPOS1	0414	OUTEEE	E1D1
OUTI	040E	OUTPUT	0233	OVER	029F	PDATA1	E07E
PDIV	03CE	READI	02A1	REMI	0353	REMO	027F
RSTBX	031F	SGNREM	00FF	SKIP	02DE	SUBI	02F1
SUBO	0264	SWAPIN	0342	TEST	0202	USERR	044B
XTEMP	00FD						

LETTERS

Dear Staff,

Congratulations on "our" new magazine. The first issue looks like a good start, I know it was more work than I can imagine!

Concerning feedback; I would prefer proven applications, or at least ones that are in the works, additional routines for FLEX, 6800 tid bits for filler, and a good joke now and then. How about fixes from SWTP that they include with current kits that were not included with earlier ones, (I will be sending one in I found out about) How about if someone calls SWTP about a problem or question, and they feel the rest of us would be inter ested in the answer, they just drop you a post card? Quick easy filler.

As for color, the advertisers might prefer it, but for me, the cover is enough.

Two venders I have had good service from, are Gimix and, World Wide Electronics.

I like the personal feeling of '68' Micro, and thanks for standing up for the 6800!

Jerry Sorrels
6266 Banner Ct.
Riverside, Ca. 92504

P.S. Tell Assistant Editor Mickey Ferguson he owes me a letter.

Jerry Sorrels
6266 Banner Ct.
Riverside, Ca. 92504

Hot MF-68 Stepper Motor Fix

On the first MF-68 Disk Systems after SWTP changed to Wangco drives, the LED on drive 0 was used to indicate power on, and head load. This not only was confusing but caused the drive 0 stepper motor to be energized all the time, and become quite hot compared to drive 1.

Later SWTP used Wangco drives that had been modified to remedy this over-heating problem, these modified drives are identified by the F revision after the drawing number, on the schematic that comes with the drives.

If your drives are revision E and the following fix was not included with your instructions, you'll probably want to add it.

On the disk controller board lift IC 4 pin 6, so that it is not connected to anything. Now drive 0 LED will only light when it's head is loaded.

I still wanted a power on indicator, so I mounted a suitable LED one inch from the left side and centered in the silver area just above the SWTP logo. I connected the LED's cathode to ground and the anode through a 330 ohm resistor to plus 5 volts. Now I don't forget to turn it off!

One other item of possible interest, Texas Instruments 2516 is pin compatable to the 5 volt 2716 that is used with the SWTP MP-R Eprom Programmer. The 2516 I used was a 650 ns.,but the programmer was still able to read and write to it. I was using the new MP-2 processor board that has a 1 MH clock.

ISLAND COMPUTERS
BOX 668
ST. JAMES CITY, FLORIDA 33956

March 16, 1979

'68' Micro Journal
Mr. Don Williams
3018 Hamill Road
Hixson, TN 37343

Dear Don,

The first issue of your (long overdue) '68' Micro Journal is indeed impressive. Congratulations. Your editoral says it all.

Due largely to your reply to the 'toy merchant', W. Green, I feel that your attitude deserves at least a two year's support, in the form of my subscription. (check enclosed)

After reading the entire journal, the following two facts seem very obvious:

1. The industry leader of the 6800 system is unmentioned. Hopefully, an oversight, soon to be corrected. I'm sure that you have noticed that all the recent 'new' products from SWTP, Gimix, Percom, and the others, are only attemps to catch up with some of the features that the users of MSI and their FD-8 system have been enjoying for more than two years. (nope, we don't sell the FD-8 - we're just users) To date, I have not seen a news release or advertisement, which even claims to equal, or approach, the features or 'useability' of the FD-8. We have the SWTP computer, and use it everyday, but when the choice of disk memory came up, FLEX was too much of a sacrifice, compaired to the FD-8.

2. While reading the '68' Micro Journal, the impression kept re-occuring that perhaps I was reading a 'TSC benefit association' communique. TSC is good at what they do (they saved SWTP's life), but then there ARE other vendors, and us readers would prefer a broader coverage. (of course, advertising budgets must be considered) However, this particular evil is surely the factor with which Wayne Green is unable to cope, in his devotion to the bottom end systems which he supports.

We've got the best system - the 6800 - and your 'mag' has got to be a winner. We're on your side. Print it on newsprint, but keep it going.

Sincerely, R. C. McKay

R. C. McKay
Island Computers

PARTICIPATORY WORKSHOP ON MICROCOMPUTER SOFTWARE

Christmas week 1979 at a quality Caribbean resort. Topics will include systems and application software as well as professional, educational and small business programs (grouped for the popular CPU's as well as available higher languages). Volunteers are needed now to help organize each area of interest. RSVP immediately for further details on this not-for profit users' holiday-workshop. Families are welcome.

Dr. Andy Bender
400 Old Hook Road
Westwood, N.J. 07675
201-664-4882 (days)
201-652-0157 (nites/weekends)

Dr. Jeff Brownstein
2 Tor Road
Wappingers Falls, N.Y. 12590
914-297-3950

** NOTICE **

All new subscribers please note the following:

As 68 Micro Journal continues to grow, back issues are becoming more and more difficult to supply. Many new

subscriptions are received requesting a start with issue number 1. Others request issue 2 as a starting issue. Due to a couple of reasons, we can not honor these request.

I understand that many would desire a full set, from issue number 1. However; issue number 1 is depleted. We are growing at a much faster pace than ever anticipated. Fact is, we are 752% ahead of where we thought we would be at this date in time.

Prior to our first issue, way back in September 1978, we commissioned a study and survey, to determine the market potential for our type publication. The results looked good, so I took the plunge. They sure missed the mark. We are better than 752 percent ahead of their predictions. This means that issue 1 backissues ran out quick.

I hope to rerun issue number 1, in the near future. The cost of such a project will be stiff. If you desire copies of issue number 1, please let us know. I need a feel of about how many to run next time around.

The policy is this......We will add all new subscriptions to the update list, going to the mailers, within one week of the next issue due out. This is much faster than other magazines. I don't know how long we can keep this up but we will for as long as we can.

Please do not request your subscription to start with a backdate issue. If you want a back issue, all except issue 1, are available. The price is $2.00 (cover price), plus $1.50 postage and handling. The postage alone, first class, is $1.06. I know this is stiff for some but it is the only way we can handle backissues.

-DMW-

TRANSFER (FLEX-1 to FLEX-2)

Dennis Womack
Rt. 4, 109 Foster Dr.
Ringgold, GA 30736

Let's say that you have just upgraded to TSC's FLEX 2.0 on your minifloppy disk system. What happens to all those files you stored on MINIFLEX disks? Can you just read them from FLEX 2.0? The answer is no. MINIFLEX sectors are 128 bytes long and FLEX 2.0 sectors are 256 bytes long. So you see, there is a problem. If you have enough memory so that FLEX 2.0 and MINIFLEX can both be in memory at once, this is a program that can help you. You must have memory from $7000 to $7FFF and from $A000 to $BFFF. To use this program, boot up your system with the FLEX 2.0 system disk. Exit by using the MON command. This will leave FLEX 2.0 in memory. Now boot up with the MINIFLEX system disk, again exit by using the MON command. This leaves MINIFLEX in memory. Re-enter FLEX 2.0 at $AD03. Insert the MINIFLEX disk containing the file you desire to transfer into drive 1. A FLEX 2.0 disk with the TRANSFER command must be placed in drive 0. Be sure you have enough room on this disk for the file you wish to transfer from the MINIFLEX disk. If you divide the number of sectors in the source file by two, this will give the maximum number of sectors in the FLEX 2.0 copy. To transfer the file, simply type TRANSFER file spec. This might look like:

```
TRANSFER FILE.TXT.1
```

The drive number is not necessary. The source file will automatically be read from drive 1 and written to drive 0.

Let's go through the operation of the program and see how it works. Each version of FLEX keeps a current track table in the disk drivers to remember the current head position for each drive. TRANSFER first patches the MINIFLEX drivers to use the same table as FLEX 2.0. We want the right hand to know what the left hand is doing.

We now get the file name of the target file, by using a routine in FLEX called GETFIL (GFILE5). The name is left in the FLEX 2.0 file control block (FCB). Then we open a FLEX 2.0 file of the same name for writing on drive 0. Putting a 2 in the first word of the FCB opens the file for writing. Calling the routine SETEXT with 1 in the a register sets the default extension to txt. Setting the fourth word in the FCB to 0 selects drive 0. Space compression is turned off so that binary as well as text files may be transfered. Placing $FF in word 59 of the FCB does this. If there are any errors we use the FLEX 2.0 error handler.

After we transfer the name of the file, a 1 is placed in the first word of the FCB. This indicates an open for read operation. These operations are the same as before, except that equivalent routines from MINIFLEX are used. The MINIFLEX file of the same name is now open for reading on drive 1. Space compression is again turned off. If we have any errors we use the MINIFLEX error handler.

Once both files are open, transfering the data is a simple matter. One byte at a time is read from the MINIFLEX file and written to the FLEX 2.0 file. Calling each file management system in turn is all it takes. This loop is exited by encountering an error. If all is working properly this is an end-of-file error. When this happens both files are closed and the program returns control to FLEX 2.0.

All errors are handled by the appropriate version of FLEX. The version encountering the error is printed to help you know where the problem is.

The reader is encouraged to buy copies of TSC'S Advanced Programmer's Guide to FLEX and Advanced Programmer's Guide to MINIFLEX. These contain a wealth of information for the system programmer.

Let's suppose you need to transfer a file back the other direction. Can you figure out how to do it?

MINIFLEX and FLEX are trademarks of Technical Systems Consultants, Inc.

DW

```
                    NAM    TRANSFER
**
* TRANSFER UTILITY
*
* THIS UTILITY RUNS UNDER FLEX 2.0 AND WILL
* COPY A MINIFLEX FILE BYTE FOR BYTE
* TO A FLEX 2.0 FILE OF THE SAME NAME.
*
* TO USE:
* 1. BOOT FLEX 2.0.
* 2. EXIT VIA "MON".
* 3. BOOT MINIFLEX.
* 4. EXIT VIA "MON".
* 5. RETURN TO FLEX 2.0 AT $AD03.
* 6. INSERT FLEX 2.0 DESTINATION DISK WITH
*    THIS PROGRAM IN DRIVE 0.
* 7. INSERT MINIFLEX DISK IN DRIVE 1.
* 8. TYPE "TRANSFER,FILE SPEC".
* 9. FILE WILL BE TRANSFERED.
*
* WRITTEN BY DENNIS WOMACK
* COPYRIGHT 2/10/79
* ALL RIGHTS RESERVED
**

*
* ROUTINES USED FROM MINIFLEX AND FLEX 2.0
*
SETEXT  EQU   $712D      MINIFLEX SETEXT
RPTERR  EQU   $713C      MINIFLEX RPTERR
WARMS2  EQU   $AD03      FLEX 2.0 WARMS
GFILE2  EQU   $AD2D      FLEX 2.0 GETFIL
SEXT2   EQU   $AD33      FLEX 2.0 SETEXT
RERR2   EQU   $AD3F      FLEX 2.0 RPTERR
PSTRNG  EQU   $AD1E      FLEX 2.0 PSTRNG
FMS     EQU   $7806      MINIFLEX FMS
FMSCLS  EQU   $7803      MINIFLEX FMSCLS
FMS2    EQU   $B406      FLEX 2.0 FMS
FMSCL2  EQU   $B403      FLEX 2.0 FMSCLS
*
* FILE CONTROL BLOCKS
*
FCB     EQU   $7740      MINIFLEX FCB
FCB2    EQU   $A840      FLEX 2.0 FCB
*
* TRANSFER UTILITY
*
0100                 ORG   $100
0100 20 01    TRNSFR BRA   START
0102 01              FCB   1         VERSION NUMBER
0103 CE BE 99 START  LDX   #$BE99    PATCH MINIFLEX DRIVERS
0106 FF 7F ED        STX   $7FED     THE SAME DRIVE
0109 CE BE 95        LDX   #$BE95    ACTIVE INFORMATION
010C FF 7F E1        STX   $7FE1     AS FLEX 2.0
010F FF 7F F0        STX   $7FF0
*
* GET TARGET FILE NAME
*
0112 CE A8 40        LDX   #FCB2     POINT TO FLEX 2.0 FCB
0115 BD AD 2D        JSR   GFILE2    GET FILE SPEC.
0118 25 74           BCS   ERROR3    BRANCH IF ERROR
*
```

```
                  * OPEN FLEX 2.0 FILE ON DRIVE 0 FOR WRITE
                  *
011A CE A8 40             LDX    #FCB2
011D 86 02                LDA A  #2         SET UP WRITE CODE
011F A7 00                STA A  0,X        SAVE OPEN FOR WRITE
0121 86 01                LDA A  #1
0123 BD AD 33             JSR    SEXT2      SET TEXT EXT
0126 86 00                LDA A  #0
0128 A7 03                STA A  3,X        SELECT DRIVE 0
012A BD B4 06             JSR    FMS2       OPEN FILE
012D 26 5F                BNE    ERROR3     BRANCH IF ERROR
012F 86 FF                LDA A  #$FF       TURN OFF SPACE COMP
0131 A7 3B                STA A  59,X
                  *
                  * OPEN MINIFLEX FILE ON DRIVE 1 FOR READ
                  *
0133 BD 01 A3             JSR    GETNAM     TRANSFER THE FILE NAME
0136 CE 77 40             LDX    #FCB
0139 86 01                LDA A  #1         SET UP READ CODE
013B A7 00                STA A  0,X
013D BD 71 2D             JSR    SETEXT     SET TEXT EXT
0140 86 01                LDA A  #1
0142 A7 03                STA A  3,X        SELECT DRIVE 1
0144 BD 78 06             JSR    FMS        OPEN FILE
0147 26 37                BNE    ERROR2     BRANCH IF ERROR
0149 86 FF                LDA A  #$FF       TURN OFF SPACE COMP
014B A7 3B                STA A  59,X
                  *
                  * TRANSFER FILE DATA
                  *
014D CE 77 40     LOOP    LDX    #FCB
0150 BD 78 06             JSR    FMS        GET CHAR FROM SOURCE
0153 26 0A                BNE    ERROR1     BRANCH IF ERROR
0155 CE A8 40             LDX    #FCB2
0158 BD B4 06             JSR    FMS2       WRITE CHAR TO DESTINATION
015B 26 31                BNE    ERROR3     BRANCH IF ERROR
015D 20 EE                BRA    LOOP       GO BACK FOR MORE
                  *
                  * MINIFLEX END-OF-FILE HANDLER
                  *
015F A6 01        ERROR1  LDA A  1,X        GET ERROR STATUS
0161 81 08                CMP A  #8         IS IT EOF?
0163 26 1B                BNE    ERROR2     NO, BRANCH
0165 CE 77 40             LDX    #FCB
0168 86 04                LDA A  #4         CLOSE FILE CODE
016A A7 00                STA A  0,X
016C BD 78 06             JSR    FMS        CLOSE MINIFLEX FILE
016F 26 0F                BNE    ERROR2     BRANCH IF CLOSE ERROR
0171 CE A8 40             LDX    #FCB2
0174 86 04                LDA A  #4
0176 A7 00                STA A  0,X
0178 BD B4 06             JSR    FMS2       CLOSE FLEX 2.0 FILE
017B 26 11                BNE    ERROR3     BRANCH IF ERROR
017D 7E AD 03             JMP    WARMS2     RETURN TO FLEX 2.0
                  *
                  * FLEX 2.0 ERROR HANDLER
                  *
0180 CE 02 12     ERROR2  LDX    #MSG1
0183 BD AD 1E             JSR    PSTRNG     PRINT "MINIFLEX"
0186 CE 77 40             LDX    #FCB
```

```
0189 BD 71 3C           JSR    RPTERR    REPORT ERROR #
018C 20 0C              BRA    ERROR4    EXIT
                 *
                 * MINIFLEX ERROR HANDLER
                 *
018E CE 02 1B    ERROR3 LDX    #MSG2
0191 BD AD 1E           JSR    PSTRNG    PRINT "FLEX 2.0"
0194 CE A8 40           LDX    #FCB2
0197 BD AD 3F           JSR    RERR2     REPORT ERROR #
019A BD 78 03    ERROR4 JSR    FMSCLS    CLOSE ALL MINIFLEX FILES
019D BD B4 03           JSR    FMSCL2    CLOSE ALL FLEX 2.0 FILES
01A0 7E AD 03           JMP    WARMS2    RETURN TO FLEX 2.0
                 *
                 *
                 *
01A3 CE A8 40    GETNAM LDX    #FCB2
01A6 A6 04              LDA A  4,X
01A8 CE 77 40           LDX    #FCB
01AB A7 04              STA A  4,X
01AD CE A8 40           LDX    #FCB2
01B0 A6 05              LDA A  5,X
01B2 CE 77 40           LDX    #FCB
01B5 A7 05              STA A  5,X
01B7 CE A8 40           LDX    #FCB2
01BA A6 06              LDA A  6,X
01BC CE 77 40           LDX    #FCB
01BF A7 06              STA A  6,X
01C1 CE A8 40           LDX    #FCB2
01C4 A6 07              LDA A  7,X
01C6 CE 77 40           LDX    #FCB
01C9 A7 07              STA A  7,X
01CB CE A8 40           LDX    #FCB2
01CE A6 08              LDA A  8,X
01D0 CE 77 40           LDX    #FCB
01D3 A7 08              STA A  8,X
01D5 CE A8 40           LDX    #FCB2
01D8 A6 09              LDA A  9,X
01DA CE 77 40           LDX    #FCB
01DD A7 09              STA A  9,X
01DF CE A8 40           LDX    #FCB2
01E2 A6 0A              LDA A  10,X
01E4 CE 77 40           LDX    #FCB
01E7 A7 0A              STA A  10,X
01E9 CE A8 40           LDX    #FCB2
01EC A6 0B              LDA A  11,X
01EE CE 77 40           LDX    #FCB
01F1 A7 0B              STA A  11,X
01F3 CE A8 40           LDX    #FCB2
01F6 A6 0C              LDA A  12,X
01F8 CE 77 40           LDX    #FCB
01FB A7 0C              STA A  12,X
01FD CE A8 40           LDX    #FCB2
0200 A6 0D              LDA A  13,X
0202 CE 77 40           LDX    #FCB
0205 A7 0D              STA A  13,X
0207 CE A8 40           LDX    #FCB2
020A A6 0E              LDA A  14,X
020C CE 77 40           LDX    #FCB
020F A7 0E              STA A  14,X
0211 39                 RTS
```

```
                          *
                          *
                          *
     0212 4D              MSG1    FCC     /MINIFLEX/
     0213 49 4E
     0215 49 46
     0217 4C 45
     0219 58
     021A 04                      FCB     4
     021B 46              MSG2    FCC     /FLEX 2.0/
     021C 4C 45
     021E 58 20
     0220 32 2E
     0222 30
     0223 04                      FCB     4
                                  END     TRNSFR

NO ERROR(S) DETECTED

  SYMBOL TABLE:

ERROR1 015F   ERROR2 0180   ERROR3 018E   ERROR4 019A   FCB    7740
FCB2   A840   FMS    7806   FMS2   B406   FMSCL2 B403   FMSCLS 7803
GETNAM 01A3   GFILE2 AD2D   LOOP   014D   MSG1   0212   MSG2   021B
PSTRNG AD1E   RERR2  AD3F   RPTERR 713C   SETEXT 712D   SEXT2  AD33
START  0103   TRNSFR 0100   WARMS2 AD03
```

A 6800 TIMING DELAY

Howard Berenbon
27200 Franklin Rd. No. 105
Southfield, MI 48034

An important tool for 'real-time' programming is the time delay. It is a necessary program for design of home applications. It can be used in burglar alarm systems, home control, games, and in any case where software timing is required.

In the case of an alarm system, the time delay can be used for pulsing the alarm bell, when a break-in is detected. If the alarm is battery operated, this will reduce battery drain. It will also keep the neighborhood annoyance to a minimum. A two minute 'on' time and a one minute 'off', with a one or two hour automatic shut-down, would be adequate.

Program delays may be used in controlling home appliances. An electric coffee pot may be interfaced to your system, through an AC control box. Using a delay as part of your program, you can enter in the amount of time required to make the desired strength of coffee. Your system will activate the AC control box, which will turn on the coffee. After a pre-determined time the coffee will be done and the pot will be turned off. Then, an alarm will be activated, indicating that your coffee is ready.

TIMING

Processor timing is important to the design of delays. The speed and number of cycles required for instruction execution is necessary for accurate design.

The 6800 operates at 1 mHz. This means that the processor cycle is 1 uS. An instruction, such as BRA, requires 2 processor cycles to be executed, or 2 uS. This information is available, for each instruction, in the M6800 Microprocessor Instruction Set Summary.

The average number of cycles per instruction is 4. A four line program may take about 20 uS to run, if there are no loops in the program.

A typical time delay subroutine is given below. It will delay 1 millisecond each time it is called. Accumulator B is used to set a count of 165 uS. Each time the program loops, from step 4 to step 2, the computer wastes 6 uS. Accumulator B is decremented, and checked for zero. When it reaches zero, the loop is broken and the 6800 executes a CMPX 0000 and an RTS. This adds the remaining time to complete the 1

millisecond delay. This subroutine may be a basis for other length delays. If it's looped 100 times, you have a 100 millisecond delay, and so on.

1 MILLISECOND DELAY

```
LINE #    ADDRESS    OBJECT CODE    MNEMONIC

0001      0100       C6 F0     MILLI LDAB F0     Set 165 count
0002      0102       5A           UP DECB        Decrement B
0003      0103       27 02           BEQ OUT     If=0, out
0004      0105       20 FB           BRA UP      Continue
0005      0107       8C 0000     OUT CPX 0000    Waste time
0006      010A       39              RTS         Return
0007                                 END

MNEMONIC          # OF CYCLES

LDAB              2
DECB              2 ----
BEQ               2 ---- = 6 uS
BRA               2 ----
CPX               3
RTS               5
```

CALCULATION

6 uS X 165 = 990 uS

990 uS + 2 uS + 3 uS + 5 uS = 1000 uS

1000 uS = 1 Millisecond

MAKE LIKE A 6809

M6809 Emulator by the Micro Works

Dr. Ted Feintuch
4188 Gann Store Rd.
Hixson, TN 37343

Amidst the excitement of a new chip, the 6809, which promises far greater computer power than the 6800, the Micro Works have produced a software package that allows prospective 6809 users a preliminary glimpse at the characteristics of the new computer.

The package includes the Emulator at three different memory locations, 3000, 6000, and C000. There is also a textfile including part of a well notated assembly listing. It contains all the documentation necessary to run the program.

Continued on page 28

To use the program, one enters 6809 code at a convenient spot in memory using the ROM monitor. If not already done, the Emulator program is then loaded in a location, in memory, not in conflict with the data.

Then cause the monitor to jump to -8A7 (- indicates 3, 6 or C). The Emulator has its own small monitor which uses an exclamation point for a prompt. Acceptable replies include "J", "S", "R", space or carriage return.

"R" causes the 6809 registers to be displayed. A "J" or "S" causes a space to be pointed to the terminal. The address of the 6809 code is typed on the terminal. If "J", the Emulator will begin to execute the code, printing a listing of all 6809 registers for each instruction. The number of instructions to be executed is determined by the value of the contents of the "A" register.

The "S" instruction causes the Emulator to single step through the 6809 code. Typing a "space" will cause execution of the next step. A carriage return will cause return to computer monitor. Illegal commands are ignored.

The Emulator is obviously not going to be used to examine elaborate program written in 6809 code. The 6809 is meant to be programmed in assemblv language because many instructions have post-instruction bytes in which each bit has a specific meaning (like the condition-code of the 6800).

For experimentation with small sequences of code and for studying the numerous new stack manipulation and addressing instructions in the 6809, the M6809 Emulator by Micro Works is an excellent tool.

Tom Harmon
HHH Enterprises
Box 493
Laurel, MD 20810

SOME BASIC GAMES

In keeping with what my accountant calls my books, I will attempt to start a Fantasy game group for the 6800. The following programs written in Computerware basic, should run in most machines without changes. They do require a disk and sequential files. They were originally developed on a cassette system but have been rewritten for disk. The first program runs very slowly and generates ten files on the disk that contain an array of numbers that represent a 10 by 10 matrix of rooms. The rooms form a maze and there is at least one path through from the top (north) to the bottom (south). If the disk already contains a set of files, it will erase and replace them. If some of the code looks clumsy, that is because there are later versions of this that do more things. By the way, the name I

call it is "maz.gen". Normally I run this only about once every 2 or 3 months, and some users have not run it a second time at all.

The second program is the actual game(?). It does not display the rules as most fantasy games require that you learn by doing. This opens by getting one of the room files at random. That will prevent memorization. Once the room file is loaded it will place the player IN the first room (without telling its' position) and prompt for a direction to travel. From there on it is a fight to get out the south side exit with whatever loot you find along the way. The subroutines that control the gold and danger statistics can easily be modified by the user to reflect his/her own sadistic inclinations. In the data line that contains I,I,,,,I,I the spaces between the commas have some acceptable control character (cntrl d) between them.

I am interested in seeing what variations turn up on this game over the months. Also be it known that there is a VERY large version coming along (not for free) that uses 32k, supports several players, (not necessarily at the same time), remembers your last move, and includes many goodies. Enough for the plug - now the listings.

```
0010 REM THIS PROGRAMME IS THE PROPERTY OF
0020 REM H H H ENTERPRISES, BOX 493, LAUREL, MD.
0030 REM ZIP 20810. ALL RIGHTS ARE RESERVED.
0040 REM USE OF THIS PROGRAMME IS FOR PERSONAL ONLY.
0050 REM THIS CANNOT BE DUPLICATED, SOLD OR GIVEN
0060 REM AWAY WITHOUT WRITTEN CONSENT OF H H H ENTERPRISES.
0070 REM THIS GENERATES, ON A DISK, 10 SETS OF
0080 REM SEQUENTIAL FILES THAT CONTAIN 100 ROOMS
0090 REM IN THE FORM OF A MAZE THAT ARE USED BY
0100 REM THE PROGRAMME "EXPLOR", (VER. 1.XXX THIS
0110 REM CANNOT BE USED WITH VERSIONS ABOVE 1.999).
0120 LINE= 0
0130 BASE= 0
0140 DIM A(11,11),B(11,11)
0150 FOR Z=1 TO 10
0160 LET G$="MAZ"+STR$(Z)+".DAT"
0170 IF FCHK G$<>0 THEN 190
0180 FDEL G$
0190 OPEN 1,G$
0200 PRINT "WORKING ON ";G$
0210 FOR I=1 TO 10:FOR J=1 TO10
0220 LET A(I,J)=0:NEXT J:NEXT I
0230 FOR I=0 TO 11:A(I,0)=-1:NEXT I
0240 FOR I=0 TO 11:A(I,11)=-1:NEXT I
0250 FOR I=0 TO 11:A(0,I)=-1:A(11,I)=-1:NEXT I
0260 FOR I=0 TO 11:FOR J=0 TO 11
0270 LET B(I,J)=3
0280 NEXT J:NEXT I
0290 LET H=10:V=10
0300 LET D=INT(RND*H)+1
0310 LET A(1,D)=1
0320 GOTO 560
0330 LET X1=INT(RND*3)-1
0340 LET A=RND
0350 LET Y1=INT(RND*3)-1
0360 IF X1=0 THEN 390
```

```
0370 IF Y1=0 THEN 390
0380 GOTO 330
0390 IF X1+X<1 THEN 330
0400 IF Y1+Y<1 THEN 330
0410 IF X1+X>V THEN 330
0420 IF Y1+Y>H THEN 330
0430 IF A(X+1,Y)=0 THEN 480
0440 IF A(X-1,Y)=0 THEN 480
0450 IF A(X,Y+1)=0 THEN 480
0460 IF A(X,Y-1)=0 THEN 480
0470 LET K=1:RETURN
0480 IF A(X+X1,Y+Y1)<>0 THEN 330
0490 IF X1=0 THEN 520
0500 IF X1>0 THEN B(X,Y)=B(X,Y)-2:GOTO 540
0510 LET B(X-1,Y)=B(X-1,Y)-2:GOTO 540
0520 IF Y1>0 THEN B(X,Y)=B(X,Y)-1:GOTO 540
0530 LET B(X,Y-1)=B(X,Y-1)-1
0540 LET A(X1+X,Y+Y1)=A(X,Y)+1:X=X+X1:Y=Y+Y1
0550 GOTO 330
0560 LET X=1:Y=D
0570 GOSUB 330
0580 LET M=0
0590 FOR I=1 TO V:FOR J=1 TO H
0600 IF A(I,J)<>0 THEN 700
0610 IF A(I+1,J)<1 THEN 630
0620 LET X=I+1:Y=J:GOTO 690
0630 IF A(I-1,J)<1 THEN 650
0640 LET X=I-1:Y=J:GOTO 690
0650 IF A(I,J-1)<1 THEN 670
0660 LET X=I:Y=J-1:GOTO 690
0670 IF A(I,J+1)<1 THEN 700
0680 LET X=I:Y=J+1
0690 LET M=1:I=V:J=H
0700 NEXT J
0710 NEXT I
0720 IF M<>0 THEN GOTO 570
0730 LET B=0
0740 FOR J=1 TO H
0750 IF A(V,J)>B THEN B=A(V,J)
0760 NEXT J
0770 FOR J=1 TO H
0780 IF A(V,J)=B THEN B(V,J)=B(V,J)-2:J=H
0790 NEXT J
0800 REM NOW WRITE TO THE DISK
0810 WRITE 1.D
0820 FOR J=1 TO 10:FOR I=1 TO 10
0830 WRITE 1,B(J,I)
0840 NEXT I:NEXT J
0850 CLOSE 1
0860 NEXT Z
0870 END
```

```
0010 REM THIS IS "EXPLOR", A GAME FROM H H H ENTERPRISES
0020 REM THIS USES 10 SEQUENTIAL FILES THAT CONTAIN
0030 REM A MAZE OF ROOMS FROM THE DISK AND PICKS ONE
0040 REM AT RANDOM TO USE
0050 REM THE OPTION OF THE WIND IS NOT IMPLEMENTED IN THIS
0060 REM VERSION. ALL RIGHTS REMAIN WITH H H H ENTERPRISES
```

```
0070 REM AND THIS GAME CANNOT BE USED FOR OTHER THAN PERSONAL
0080 REM USE AND CANNOT BE SOLD OR GIVEN AWAY WITHOUT PERMISSION
0090 REM OF H H H ENTERPRISES IN WRITING.
0100 GOTO 150
0110 FOR Z1=1 TO 5:NEXT Z1
0120 RETURN
0130 PRINT A$;
0140 RETURN
0150 LINE= 0
0160 LET D$="MAZ"
0170 DIM A$(12)
0180 DIM O$(5),O(5)
0190 FOR I=1 TO 5
0200 READ O$(I),O(I)
0210 NEXT I
0220 LET M=0
0230 LET H3=10
0240 LET E=0
0250 FOR I=1 TO 7
0260 READ A$(I)
0270 NEXT I
0280 FOR I=3 TO 5
0290 LET A$(I)=" "
0300 NEXT I
0310 DIM D(10,10)
0320 LET X=INT(RND(0)*10)+1
0330 LET G$=D$+STR$(X)+".DAT"
0340 OPEN 1,G$
0350 READ 1,X9 : REM ENTRANCE DOOR LOCATION
0360 FOR V=1 TO 10
0370 FOR H=1 TO 10
0380 READ 1,D(V,H)
0390 NEXT H
0400 NEXT V
0410 CLOSE 1
0420 GOTO 590
0430 PRINT "I------       -----I"
0440 RETURN
0450 PRINT "I------------------I"
0460 RETURN
0470 LET X7=1:IF INT(D(V1,H1))<>D(V1,H1) THEN X7=2
0480 FOR I=1 TO 7
0490 ON X8 GOSUB 570,550
0500 PRINT TAB(20);
0510 ON INT(D(V1,H1))+1 GOSUB 570,550,570,550
0520 PRINT
0530 NEXT I
0540 RETURN
0550 PRINT "I";
0560 RETURN
0570 PRINT A$(I);
0580 RETURN
0590 GOSUB 2480
0600 LET V1=1:H1=X9
0610 PRINT "YOU ARE STANDING IN THE "
0620 PRINT "ENTRANCE TO A CASTLE'S DUNGEONS."
0630 PRINT "TO SPECIFY A DIRECTION TYPE:"
0640 PRINT "N= NORTH (UP)"
0650 PRINT "S= SOUTH (DOWN)"
0660 PRINT "E= EAST  (RIGHT)"
```

```
0670 PRINT "W= WEST  (LEFT)"
0680 PRINT "R= REST  (STAND STILL)"
0690 PRINT "YOUR TORCH ONLY ILLUMINATES"
0700 PRINT "AN AREA OF APROX. 10' BY 10'."
0710 PRINT "WATCH OUT FOR THE WIND!"
0720 PRINT "WHEN READY HIT RETURN.";
0730 INPUT Z$
0740 GOTO 1110
0750 IF V1=1 THEN IF H1=X9 THEN GOSUB 430
0760 IF V1=1 THEN IF H1<>X9 THEN GOSUB 450
0770 IF V1>1 THEN 800
0780 GOTO 840
0790 GOTO 840
0800 IF INT(D(V1-1,H1))<2 THEN 830
0810 GOSUB 450
0820 GOTO 840
0830 GOSUB 430
0840 LET X8=1
0850 IF H1=1 THEN X8=2
0860 IF H1>1 THEN 890
0870 GOSUB 470
0880 GOTO 920
0890 IF D(V1,H1-1)=1 THEN X8=2
0900 IF D(V1,H1-1)=3 THEN X8=2
0910 GOSUB 470
0920 ON INT(D(V1,H1))+1 GOTO 430,430,450,450
0930 PRINT
0940 PRINT "WHAT DIRECTION";
0950 INPUT Z$
0960 FOR Z1=1 TO 8
0970 IF Z$="N" THEN 1030
0980 IF Z$="S" THEN 1050
0990 IF Z$="E" THEN 1070
1000 IF Z$="W" THEN 1090
1010 IF Z$="R" THEN 1100
1020 GOTO 940
1030 LET V2=-1
1040 RETURN
1050 LET V2=1
1060 RETURN
1070 LET H2=1
1080 RETURN
1090 LET H2=-1
1100 RETURN
1110 REM THIS IS START OF DISPLAY
1120 GOSUB 130
1130 LET V2=0
1140 LET H2=0
1150 GOSUB 2240
1160 GOSUB 750
1170 GOSUB 930
1180 IF V1+V2<1 THEN 2070
1190 IF V1+V2>10 THEN 1500
1200 IF H1+H2<1 THEN 1370
1210 IF H1+H2>10 THEN 1370
1220 IF V2<>0 THEN 1290
1230 IF H2<>0 THEN 1330
1240 GOSUB 1740
1250 IF H3<=0 THEN 1650
1260 GOSUB 2120
```

```
1270 GOSUB 2090
1280 GOTO 1110
1290 LET X6=0:IF V2=-1 THEN X6=-1
1300 IF INT(D(V1+X6,H1))>=2 THEN 1370
1310 LET V1=V1+V2
1320 GOTO 1240
1330 LET X6=0:IF H2=-1 THEN X6=-1
1340 IF INT(D(V1,H1+X6))<>0 THEN IF INT(D(V1,H1+X6))<>2 THEN 1370
1350 LET H1=H1+H2
1360 GOTO 1240
1370 PRINT "HIT YOUR HEAD ON WALL"
1380 PRINT "HEALTH SUFFERS '-1'."
1390 LET H3=H3-1
1400 GOSUB 2480
1410 GOTO 1240
1420 PRINT "THIS IS THE ENTRANCE"
1430 PRINT "ARE YOU SURE YOU WANT TO "
1440 PRINT "LEAVE";
1450 INPUT Z$
1460 IF LEFT$(Z$,1)="Y" THEN 1590
1470 LET V2=0
1480 GOSUB 930
1490 GOTO 1180
1500 IF INT(D(V1,H1))<>3 THEN IF INT(D(V1,H1))<>1THEN 1370
1510 PRINT "THIS IS THE EXIT."
1520 PRINT "ARE YOU SURE YOU WANT TO"
1530 PRINT "LEAVE";
1540 INPUT Z$
1550 IF LEFT$(Z$,1)="Y" THEN 2300
1560 LET V2=0
1570 GOSUB 930
1580 GOTO 1180
1590 REM
1600 IF E>0 THEN 1620
1610 PRINT "C H I C K E N ! !"
1620 GOSUB 2240
1630 PRINT "BETTER LUCK NEXT TIME!"
1640 GOTO 2400
1650 REM
1660 GOSUB 2240
1670 PRINT
1680 PRINT "TOO BAD!!!!!!!!!"
1690 PRINT "YOU ARE DEAD ! ! ! !"
1700 LET M=0
1710 GOTO 2400
1720 DATA GARGOYLE,2,DRAGON,4,WARLOCK,3,TROLL,1,TAX COLLECTOR,2
1730 DATA I,I,,,,I,I:REM THE NONPRINTING CHAR. ARE CNTRL D
1740 IF Z$="R" THEN 1770
1750 IF RND(0)<.08 THEN 1790
1760 RETURN
1770 IF RND(0)<.2 THEN 1790
1780 RETURN
1790 PRINT "SURPRISE! A ";
1800 LET R=INT(RND(0)*5+1)
1810 PRINT O$(R)
1820 PRINT "WANTS TO FIGHT."
1830 PRINT "DO YOU WANT TO FIGHT";
1840 INPUT Z$
1850 IF LEFT$(Z$,1)="Y" THEN 1960
1860 PRINT "YOU ARE RUNNING AWAY AND"
```

```
1870 PRINT "GET LOST.....OR KILLED...."
1880 LET V1=INT(RND(0)*10+1)
1890 LET H1=INT(RND(0)*10+1)
1900 LET H2=0
1910 LET V2=0
1920 LET H3=H3-1
1930 IF RND(0)<.077 THEN H3=0
1940 GOSUB 2480
1950 RETURN
1960 REM
1970 PRINT "BE BRAVE - THE FIGHT IS ON - ---"
1980 PRINT "----------------------------------"
1990 IF RND(0)>.97 THEN 2030
2000 PRINT "YOU WON!! BUT HAVE SOME DAMAGE."
2010 LET H3=H3-O(R)
2020 RETURN
2030 PRINT "YOU LOST. HEAVY DAMAGE TO YOUR"
2040 PRINT "HEALTH. YOU LOST ";(O(R)*2);"POINTS."
2050 LET H3=H3-(O(R)*2)
2060 RETURN
2070 IF H1<>X9 THEN 1370
2080 GOTO 1420
2090 IF H3>9 THEN RETURN
2100 LET H3=H3+.25
2110 RETURN
2120 IF RND(0)>.8 THEN 2140
2130 RETURN
2140 PRINT "SURPRISE!! YOU FOUND GOLD!"
2150 PRINT "WORTH ";
2160 LET L=INT(RND(0)*100+10)
2170 IF RND(0)>.92 THEN 2220
2180 PRINT L
2190 LET M=M+L
2200 GOSUB 2480
2210 RETURN
2220 LET L=INT(RND(0)*10E4+1E3)
2230 GOTO 2180
2240 PRINT "YOUR HEALTH IS";
2250 PRINT H3
2260 PRINT "YOUR CASH VALUE IS $";M
2270 PRINT
2280 RETURN
2290 REM
2300 GOSUB 2240
2310 IF M>10000 THEN 2350
2320 IF M>1000 THEN 2380
2330 PRINT "CHICKEN!!!!!!"
2340 GOTO 2400
2350 PRINT "GOOD JOB! NOW GO PUT A DOWN"
2360 PRINT "PAYMENT ON A CASTLE."
2370 GOTO 2400
2380 PRINT "NOT MUCH $ BUT AT LEAST YOU"
2390 PRINT "LIVED."
2400 PRINT "DO YOU WANT TO TRY AGAIN";
2410 INPUT Z$
2420 IF LEFT$(Z$,1)="Y" THEN 2450
2430 IF LEFT$(Z$,1)="N" THEN END
2440 GOTO 2400
2450 LET H3=10
2460 GOTO 590
2470 END
2480 FOR Z=1 TO 100
2490 NEXT Z
2500 RETURN
```

A STAFF REVIEW
The 68 Micro Journal Lab

One item often needed, but not always available is a way to point-to-point trouble shoot a SS 50 bus, and not bend or disturb other system boards. Due to the close spacing (especially a well filled system) of the slots of the motherboard it is next to impossible to get scope or meter leads to the proper place, without disturbing something else.

The 68 Micro Journal lab received, for evaluation, a set of Transition Enterprises, Inc.'s 'Extender Boards'. The set is two boards, one 30 pin and the other 20 pin. This allows the extension of either set of slots, by using both on the 50 pin slots. They come complete with a full set of connectors, both top and bottom, allowing simple construction and use. Total construction time: 20 minutes.

The quality of the fiberglass boards is excellent and the foil runs wide (low noise). The connectors are 'Molex' (TM) as used on other SS 50 bus devices. The price is certainly right; $19.95. They may be ordered from:
Transition Enterprises, Inc. Star Route Box 241, Buckeye, AZ, 85326

A 68 Micro Journal lab rating of AAA.

Rating scale:
AAA - Excellent
AA - Good
A - Fair (could be better but works)
P - Poor and may not work
X - Not recommended for children (or anything else)

PAC Dale L. Puckett
163 Farm Acre Rd.
Syracuse, NY 13210

BOOT (FLEX-BFD)

Here is a routine which can be used by readers using the BFD to FLEX conversion detailed in Issue Number One of '68' Micro Journal. It allows the user to boot directly into the FLEX operating system from a cold start by typing, "J 8020."

To make this addition it is only necessary to assemble the source code accompanying this article. After it is assembled the resulting binary file must be appended to the NEWDISK1.CMD file that was described in Issue Number One.

If the user calls the appended file, NEWDISK2.CMD, and retains NEWDISK1.CMD, he will indeed have the best of both worlds.

With NEWDISK1.CMD he can create a disk which may be booted by any SWTPC MF-68 user. Conversely, with NEWDISK2.CMD, he will be able to create a disk which may be booted by any BFD-68 user.

It should be noted that after the new disk is initialized with either NEWDISK1.CMD or NEWDISK2.CMD, the DOS.SYS file must be copied to the disk and LINKed to the boot with the LINK utility. Of course, if you have a special program that does not require the DOS.SYS for operation, you could LINK to it instead. Use your imagination.

It should be noted that I placed the boot loader I/O buffer in low memory so that it would not write over the first part of the mini-FLEX system which starts at $7080.

One final thought. Users who wish to use my BFD to FLEX conversions on a system that utilizes interrupts will want to set the interrupt flag when they enter a disk operation. I did not do this in the source in the original listing. Here is a listing of the READ routine patch as adopted for use with interrupts.

```
READ NOP
     SEI
     STA A BFDTRG
     STA B BFDSRG
     STX BFDSBU
     JSR BFDRED
     NOP
     CLI
     RTS
```

The write and verify routine also require this change for operation with interrupts. Readers will find complete details to the conversion in '68' Micro Journal, Issue Number One. I hope you find the new boot routines a helpful addition to my original conversion.

```
        NAM    BOOT

    * by Dale L. Puckett
    * Chief Photojournalist
    * U. S. Coast Guard

    * 163 Farm Acre Road
    * Syracuse, New York  13210

    * April 15, 1979
```

```
* The following routines can be
* assembled.  Then, the resulting binary
* file may be appended to
* the NEWDISK1.CMD prepared as suggested
* on Page 15 of Issue Number one of
* '68' Micro Journal.

* The new file created by the append can be called NEWDISK2.CMD.
* If this is done then, NEWDISK1.CMD will create a
* disk which can be booted by an MF-68 system.

* NEWDISK2.CMD however, will create a disk which
* can be booted by any BFD-68 system.

* To boot a disk initialized by NEWDISK2.CMD
* a BFD-68 user needs only to type J 8020.
* This will bring up the FLEX system.

0534                      ORG    $534
0534 8E A0 79   BOOT      LDS    #$A079
0537 20 38                BRA    START
0539 00                   FCB    0,2
053A 02

053B CE 10 C8   READ      LDX    #$10C8    place buffer in low memory
053E B6 70 05             LDA A  $7005     get track number
0541 F6 70 06             LDA B  $7006     and Sector Number
0544 FF A0 7E   READ1     STX    $A07E     point to buffer
0547 B7 A0 7C             STA A  $A07C     and track
054A F7 A0 7D             STA B  $A07D     and sector
054D BD 80 29             JSR    $8029     BFD-68 Read routine
0550 26 E2                BNE    BOOT      If errors keep trying
0552 CE 10 CC             LDX    #$10CC
0555 FF 70 0B   READ3     STX    $700B
0558 39                   RTS

0559 FE 70 0B   READ4     LDX    $700B
055C 8C 11 48             CPX    #$1148    end of buffer?
055F 27 05                BEQ    READ5     yes, go
0561 A6 00                LDA A  0,X       else, get next character
0563 08                   INX              and bump pointer
0564 20 EF                BRA    READ3     and loop till done
0566 CE 10 C8   READ5     LDX    #$10C8    buffer
0569 A6 00                LDA A  0,X       get new track pointer
056B E6 01                LDA B  1,X       and new sector pointer
056D 8D D5                BSR    READ1     and go read it
056F 20 E8                BRA    READ4

0571 BD 80 38   START     JSR    $8038     BFD-68 Restore
0574 8D C5                BSR    READ      go read first sector in
0576 8D E1      READ6     BSR    READ4
0578 81 02                CMP A  #$02      start of file?
057A 27 13                BEQ    READ7     yes
057C 81 16                CMP A  #$16      reached transfer address?
057E 26 F6                BNE    READ6     no keep reading
0580 8D D7                BSR    READ4     get transfer address from buffer
0582 B7 70 07             STA A  $7007     and store it
0585 8D D2                BSR    READ4     get rest of it
0587 B7 70 08             STA A  $7008     and store it
058A FE 70 07             LDX    $7007     point to it
```

```
058D 6E 00               JMP    0,X      and go execute it (FLEX).

058F 8D C8        READ7  BSR    READ4
0591 36                  PSH A
0592 8D C5               BSR    READ4
0594 33                  PUL B
0595 B7 70 0A            STA A  $700A
0598 F7 70 09            STA B  $7009
059B 8D BC               BSR    READ4
059D 16                  TAB
059E 27 D6               BEQ    READ6
05A0 37           READ8  PSH B
05A1 8D B6               BSR    READ4
05A3 33                  PUL B
05A4 FE 70 09            LDX    $7009
05A7 A7 00               STA A  0,X
05A9 08                  INX
05AA FF 70 09            STX    $7009
05AD 5A                  DEC B
05AE 26 F0               BNE    READ8
05B0 20 C4               BRA    READ6

                         END

NO ERROR(S) DETECTED

  SYMBOL TABLE:
BOOT    0534   READ    053B   READ1  0544   READ3  0555   READ4  0559
READ5   0566   READ6   0576   READ7  058F   READ8  05A0   START  0571
```

FREEZE DISPLAY — SSB

Dan Johnson
Solar Computer Systems
7655 SW Cedarcrest ST.
Po lland, OR 97223

HERE IS THE LISTING OF A USEFUL PATCH TO THE SMOKE SIGNAL BROADCASTING DISK SYSTEM. IT CAUSES A CONTROL/S TYPED ON THE CONSOLE KEYBOARD TO FREEZE THE CONSOLE DISPLAY UNTIL EITHER A CONTROL/Q OR A CONTROL/C IS TYPED. THE CONTROL/Q WILL RESUME THE DISPLAY WHERE IT LEFT OFF AND THE CONTROL/C WILL ABORT THE PROGRAM GENERATING THE DISPLAY AND RETURN TO THE OPERATING SYSTEM. IF YOU ARE A NEW USER OF THE SSB DISK SYSTEM, YOU MAY BE WONDERING HOW TO STOP THE DIRECTORY LISTING FROM GOING OF THE TERMINAL SCREEN BEFORE YOU CAN READ IT. THIS PATCH WILL SOLVE THAT PROBLEM, AND IT WILL ALSO WORK FOR ALL TRANSIENTS OR OTHER PROGRAMS, (BASIC FOR EXAMPLE) THAT VECTOR THEIR CONSOLE I/O THROUGH THE OPERATING SYSTEM.

NOTE IN THE ASSEMBLY LISTING THAT THE PATCH ASSUMES THE CONTROL PORT FOR THE CONSOLE IS AN ACIA STARTING AT $B004 (PORT 1) AND THE ROM MONITOR CHARACTER OUTPUT ROUTINE STARTS AT $E1D1. IF THESE ADDRESSES ARE NOT CORRECT THEY WILL HAVE TO BE CHANGED. THE PATCH WILL NOT WORK WITH MIKBUG OR ANY OTHER MONITOR THAT USES A PIA FOR THE CONTROL PORT.

THE QUICKEST WAY TO INSTALL THE PATCH IS TO TYPE IN THE HEX CODE FROM THE LISTING USING THE ROM MONITOR MEMORY CHANGE FUNCTION, THEN SAVE THE CODE WITH THE DOS68 "SAVE" COMMAND AND "APPEND" IT TO THE OPERATING SYSTEM.

```
E.G.    SAVE,PAT1,7286,7288<RET>
        SAVE,PAT2,67C0,67F3<RET>
        APPEND,PAT1,DOS68.42<RET>
        APPEND,PAT2,DOS68.42<RET>
```

```
2:          NAM     PATCH1
3:          WITH    WI=80
```

```
                   *************************************************************
                   *PATCH FOR SSB DISK SYSTEM TO ALLOW ^S TO STOP TERMINAL OUT
                   *^Q TO RESTART IT, AND ^C TO EXIT TO OPERATING SYSTEM (WARM
                   *By: Dan Johnson
                   *     Solar Computer Systems Corp.
                   *     7655 S.W. Cedarcrest St.
                   *     Portland OR  97223
                   *************************************************************

 7283              ZWARMS  EQU     $7283
 7286              OUTEEE  EQU     $7286
 E1D1              OUTCHAR EQU     $E1D1               *THESE ADDRESSES MAY
 8004              ACIASR  EQU     $8004               *REQUIRE CHANGE....
 8005              ACIADR  EQU     $8005               *DEPENDING ON YOUR SYSTEM M
 0013              CNTRLS  EQU     $13
 0011              CNTRLQ  EQU     $11
 0003              CNTRLC  EQU     $3

 7286                      ORG     OUTEEE
7286 7E67C0                JMP     NEWOUT              PATCH IN NEW OUTPUT VECTOR

 67C0                      ORG     $67C0

                   *AFTER PRINTING EACH LINE..GO CHECK THE CONSOLE KEYBOARD
67C0 36            NEWOUT  PSH A                       SAVE CHARACTER
67C1 BDE1D1                JSR     OUTCHAR             OUTPUT CHAR. TO CONSOLE
67C4 32                    PUL A
67C5 810A                  CMP A   #$0A                WAS IT A LF?
67C7 26EA                  BNE     NOPE                IF NOT THEN RTS
                   *                                   ELSE..FALL THRU

                   *CHECK KEYBOARD TO SEE IF A CHAR. HAS BEEN TYPED...
                   *IF SO...SEE IF IT'S A ^C ^S OR ^Q
                   *IF NOT THEN JUST IGNORE IT

67C9 36            KEYCHK  PSH A                       SAVE REG
67CA B68004                LDA A   ACIASR              CHECK ACIA STATUS
67CD 47                    ASR A
67CE 2422                  BCC     KEYOUT              IF NOTHING TYPED

67D0 B68005                LDA A   ACIADR              GET DATA
67D3 8103                  CMP A   #CNTRLC             IS IT A ^C
67D5 2606                  BNE     KEY1

                   *EXIT TO OPERATING SYSTEM
67D7 32            KEYEXIT PUL A               RESTORE REG
67D8 31                    INS
67D9 31                    INS                         FIX STACK POINTER
67DA 7E7283                JMP     ZWARMS              GO TO OPERATING SYSTEM

67DD 8113          KEY1    CMP A   #CNTRLS             WAS IT A ^S
67DF 2611                  BNE     KEYOUT              IF NOT THEN IGNORE

                   *WAIT FOR A ^Q OR A ^C TO BE TYPED
67E1 B68004        KEYWAIT LDA A   ACIASR              CHECK STATUS AGAIN
67E4 47                    ASR A
67E5 24FA                  BCC     KEYWAIT             WAIT FOR SOMETHING
67E7 B68005                LDA A   ACIADR              GET CHAR.
67EA 8103                  CMP A   #CNTRLC             IS IT A ^C ?
67EC 27E9                  BEQ     KEYEXIT             IF SO...GO EXIT
67EE 8111                  CMP A   #CNTRLQ             IS IT A ^Q ?
67F0 26EF                  BNE     KEYWAIT             IF NOT GO TRY AGAIN

67F2 32            KEYOUT  PUL A                       RESTORE REG
67F3 39            NOPE    RTS
                           END
```

The program listing is written for the SWTPCo MP-68 system, but may be used with other systems with a small amount of modification. A paper tape reader is connected to port B of a parallel interface in Slot 2 of the motherboard. The PIA is programmed for a low to high transition for the "handshaking" lines.

When the paper tape reader senses a sprocket hole, the small critter, a flip-flop or one-shot will send a low to high pulse to CB1 of the board. When the program reads the data it will cause the PIA to send back an acknowledge signal on CB2 to clear the reader and get ready for the next character. This signal may be ignored by some readers such as the RAECO and PROKO readers. I use the Oliver Audio Reader, i.e. rarely now that everyone has gone from the paper tape stage to cassette and disk systems, and wired CB1 to RDA and CB2 to $\overline{ACK}$.

If an error occurs during the reading of the tape, a software interrupt will occur at location $2FC6. This will stop the program, the user can back up the tape and try again the last few records. If no error occurs, then control will return to the monitor when an S9 record is encountered.

The program is relocatable, since relative branch instructions were used for subroutine calls, and scratch locations are in the scratch area of the system monitor.

The program is years old, but may still be of use to a few readers and I still have a large quantity of paper tape stored somewhere, but saved everything to disk (PERCOM) and have not had use for them in some time.

Any questions or problems should be directed to Santa Claus, North Pole, Zip 00001.

```
                 *
                 *     PAPER TAPE READER PROGRAM
                 *
                 *     WRITTEN FOR THE SWTPCO M6800 SYSTEM
                 *
                 *     THE PROGRAM IS RELOCATABLE AND AS SHOWN RESIDES
                 *     IN THE UPPER PORTION OF 12K.  ALL SCRATCH AREA
                 *     IS IN THE MC6810 IN THE SYSTEM LOCATED $A000-$A07F.
                 *
                 *     DR. CHUCK ADAMS
                 *
                 *
                 *
            A010 CKSM   EQU   $A010    CHECKSUM SCRATCH
            A011 BYTECT EQU   $A011    BYTE COUNT PER RECORD
            A012 XHI    EQU   $A012    HIGH ORDER BYTE FOR ADDRESS GENERATION
            A013 XLOW   EQU   $A013    LOW ORDER BYTE FOR ADDRESS GENERATION
            E0D0 HUMBUG EQU   $E0D0    MONITOR ENTRY POINT
            800B PIACTL EQU   $800B    PARALLEL PORT AT 2 WITH READER
            800A PIADAT EQU   $800A    DATA REGISTER FOR PIA
```

```
                  *
2F8E                      ORG    $2F8E    GOOD START ADR FOR 12K SYSTEM
                  *
2F8E 86 2E        START   LDAA   #$2E     CONTROL BYTE FOR PIA
2F90 B7 800B              STAA   PIACTL   STORE IN CONTROL REG
2F93 B7 800A              STAA   PIADAT   STORE INTO DATA REG TO CLR IT
2F96 8D 2F        LOAD    BSR    INCH     GET CHAR FROM READER
2F98 81 53                CMPA   #'S      LOOKING FOR 'S' FOR START OF RECORD
2F9A 26 FA                BNE    LOAD     GO BACK FOR NEXT CRITTER
2F9C 8D 29                BSR    INCH     READ NEXT CHAR AFTER 'S'
2F9E 81 39                CMPA   #'9      SEE IF END OF FILE
2FA0 26 03                BNE    LOAD1    NO. GO ON
2FA2 7E E0D0              JMP    HUMBUG   END OF FILE. RETURN TO MONITOR
                  *
2FA5 81 31        LOAD1   CMPA   #'1      SEE IF DATA RECORD
2FA7 26 ED                BNE    LOAD     NOT DATA RECORD. MUST BE GARBAGE
2FA9 7F A010              CLR    CKSM     CLR CHECKSUM TO GET SERIOUS
2FAC 8D 35                BSR    BYTE     READ IN BYTE COUNT
2FAE 80 02                SUBA   #2       SUBTRACT TWO FOR COUNT
2FB0 B7 A011              STAA   BYTECT   STORE REMAINING COUNT
                  *
                  *       CREATE ADDRESS FROM NEXT FOUR CHARACTERS
                  *
2FB3 8D 20                BSR    BADDR
                  *
                  *       STORE DATA INTO MEMORY AS WE GO
                  *
2FB5 8D 2C        LOAD2   BSR    BYTE     GET DATA
2FB7 7A A011              DEC    BYTECT   SHOW WE GOT IT
2FBA 27 05                BEQ    LOAD3    WE NOW HAVE CKSUM IF ZERO COUNT
2FBC A7 00                STAA   0.X      STORE DATA IN MEMORY
2FBE 08                   INX             POINT TO NEXT LOCATION
2FBF 20 F4                BRA    LOAD2    GET MORE DATA
2FC1 7C A010      LOAD3   INC    CKSM     MAKE TWO'S COMPLEMENT
2FC4 27 D0                BEQ    LOAD     CHECKS OK. GO GET NEXT RECORD
2FC6 3F                   SWI             NO!  SORRY BUT ERROR OCCURRED.
                  *
2FC7 B6 800B      INCH    LDAA   PIACTL   GET STATUS OF READER
2FCA 2A FB                BPL    INCH     WAIT UNTIL READER SENSES NEW CHARACTER
2FCC B6 800A              LDAA   PIADAT   GET BYTE FROM READER
2FCF 84 7F                ANDA   #$7F     REMOVE HIGH ORDER BIT. MAY BE STUPID PARITY
2FD1 B7 800A              STAA   PIADAT   RESET PIA (THIS WORKS WELL)
2FD4 39                   RTS
                  *
2FD5 8D 0C        BADDR   BSR    BYTE     GET BYTE
2FD7 B7 A012              STAA   XHI      GOT HIGH ORDER BYTE OF ADDRESS
2FDA 8D 07                BSR    BYTE     GET BYTE
2FDC B7 A013              STAA   XLOW     SAVE LOW ORDER BYTE OF ADDRESS
2FDF FE A012              LDX    XHI      LOAD ALL 16 BITS INTO IX REGISTER
2FE2 39                   RTS
                  *
2FE3 8D 10        BYTE    BSR    INHEX    GET HEX CHARACTER
2FE5 48                   ASLA
2FE6 48                   ASLA
2FE7 48                   ASLA
2FE8 48                   ASLA            NOW HAVE 4 BITS IN HIGH ORDER NYBBLE
2FE9 16                   TAB             SAVE AWHILE IN B REGISTER
2FEA 8D 09                BSR    INHEX    GET NEXT HEX CHARACTER
2FEC 1B                   ABA             CREATE COMPLETE BYTE
2FED 16                   TAB             CLONE IN REGISTER B
```

```
2FEE FB A010        ADDB  CKSM    ADD TO CHECKSUM
2FF1 F7 A010        STAB  CKSM    UPDATE CHECKSUM
2FF4 39             RTS
               *
2FF5 8D D0    INHEX BSR   INCH    GET CHARACTER FROM READER
2FF7 80 30          SUBA  #$30    STRIP OFF HIGH ORDER NYBBLE
2FF9 81 09          CMPA  #$09    SEE IF WAS A-F
2FFB 2F 02          BLE   IN1HG   NO, GO BACK
2FFD 80 07          SUBA  #7      WAS A-F.  FIXIT RIGHT.
2FFF 39       IN1HG RTS
               *
               *    THAT'S ALL FOLKS
               *
                    END

SYMBOL TABLE

CKSM    A010     BYTECT A011     XHI     A012     XLOW    A013
HUMBUG  E0D0     PIACTL 800B     PIADAT  800A     START   2F8E
LOAD    2F96     LOAD1  2FA5     LOAD2   2FB5     LOAD3   2FC1
INCH    2FC7     BADDR  2FD5     BYTE    2FE3     INHEX   2FF5
IN1HG   2FFF

   0  ERRORS
>
```

The following are extracted from: FLEX NEWSLETTER No. 1, Technical Systems Consultants, Inc., PO Box 2574, West Lafayette, Indiana 47906, Copyright (c) 1979. Dated February 1979.

FULL DIRECTORY PROBLEM IN MINI FLEX

There is a problem in mini FLEX (the version supplied with the MF-68 disk system from SWTPc) which does not occur often, but is not too pleasant when it does. It occurs when you have filled the directory of a disk (exactly 75 files), then delete one or more of those files, and then try to write another file to the disk. This will crash the directory on the disk and thus cause you to lose data. Unfortunately, there is no simple patch to fix this problem. The best solution is to just be aware of it and not get into such a situation. If you get a directory full error, be SURE not to attempt to delete a file. The disk will be OK to use for reading files, but cannot be used for any other purpose since trying to write to it will give you a directory full error and trying to delete a file will give you the fatal opportunity to write to the disk which will cause the directory to be lost. It's best not to perform a delete at all. As stated, if you only wish to read from the disk, there is no problem and the disk could be used for such purposes forever. However, to make things safer (so that you don't unintentionally perform a delete and write) you should copy the files from the disk to two other disks (so that their directories aren't full) and then reformat or "NEWDISK" the one with the full directory. If you don't need all the files on the disk with a full directory, simply copy those you want to a single other disk and then reformat the filled one.

This problem does not exist in FLEX 1.0 or FLEX 2.0. In fact, the number of files in these versions is not limited to 75.

Replace the instruction at $7635 (JSR OUTADR)
with the following instructions
JSR OUTHEX
INX
JSR OUTHEX

The PDEL Utility - Volume 5, Number 2

This problem is in both versions of the utilities and the fix is the same for each:

Change the instruction at $029B in the mini FLEX
version or $A29E in the 8" FLEX 1.0 version
from BEQ DDDLR5
to BNE DDDLR6

The FIND Utility - Volume 1, Number 1

The find utility does not always perform an FMS close function before returning to warm start. This can cause problems if using the FIND utility in an EXEC command. The problem is in both the mini FLEX and 8" FLEX 1.0 versions. The fixes are not the same in both cases. First for the mini FLEX version:

Change the instruction at $01B7 from FIND35 JMP WARMS
to FIND35 JMP ERROR8

For the large disk FLEX 1.0:

Change the instruction at $A17F from JMP WARMS
to JMP FIND35
Then change the instruction at $A195
from FIND35 JMP WARMS
to FIND35 JSR FMSCLS
followed by JMP WARMS

The MAP Utility - Volume 3, Number 1

The mini FLEX version of MAP has a bug which can be fixed by replacing the routine called "PRTEND" at $01F5 thru $0201 with the following:

```
PRTEND  LDX   PREV      GET ADDRESS
        DEX             DECREMENT IT
        STX   PREV      SAVE BACK OUT
        LDX   #PREV     GET PREVIOUS ADDRESS
        JSR   OUTHEX    OUTPUT IT
        INX
        JSR   OUTHEX
        RTS
```

This change has already been made in the 8" FLEX 1.0 version of MAP.

NEW PRODUCTS

PERCOM INTRODUCES GENERAL LEDGER SYSTEM SOFTWARE

Garland, Texas - April 9, 1979 - Harold Mauch, president of Percom Data Company, announced here today the release and immediate availability of an accounting/bookkeeping software system for 6800 microcomputers.

Called Percom General Ledger System, the programs run on computers using the company's LFD-400™ dual-drive mini-disk storage device.

Mauch said the General Ledger System, together with hardware, give small-business owners a business computer system comparable to IBM's 5110 for less than one-third the cost.

Developed by a Certified Public Accountant, the Percom GLS may be operated by a secretary with little or no knowledge of bookkeeping. Moreover, the business owner needs only a superficial knowledge of computors.

Features of the Percom General Ledger System include:

* Efficient operation: Account balances are updated immediately in real time -- time-consuming sorting/posting data processing is unnecessary. Financial statements may be printed immediately after journal entries.
* Adaptability: User selects and assigns own account numbers, and formats financial statements tailored to firm's particular requirements.
* Easy-to-use: The GLS programs guide the operator throughout an application by prompting for user response with characters and non-technical questions.
* Error detection: The GLS signals the operator if journal entries do not balance. Invalid account numbers are rejected.
* Audit trail: A Posting Analysis report program provides a complete audit trail to source documents.

The Percom General Ledger System accommodates up to 250 accounts. The General Ledger System runs under Percom's Super BASIC, and the two programs together require 24K bytes of RAM.

Minimum hardware required -- in addition to a 6800 computer and LFD-400™ drive system -- is a CRT console such as the Lear-Siegler ADM-3 and a line printer capable of printing on 8½ x 11 inch paper.

The Percom General Ledger System software is supplied on mini-diskette along with a user's manual for $199.95.

The disks and manual may be ordered by dialing Percom's toll-free ordering number: 1-800-527-1592. Orders may be paid by check or money order, or charged to Visa or Master Charge credit accounts. Texas residents must add 5% sales tax.

SS 50 BUS-6800 MONITORS

USER NEED	FEATURES	GMXBUG V3.0	GMXBUG V2.0	GMXBUG V1.0	SWTBUG	RT/68	MIKBUG	SMARTBUG	MSIBUG
MONITOR SIZE	SIZE	3K	3K	2K	1K	1K	½K	1K	1K
VIDEO BASED FORMAT	FORMAT	80x24	64x16	64x16					
MEMORY EXAMINE AND CHANGE MODE	EXAMINE PREVIOUS LOCATION	—	—	—	↑			U	[illegible]
	EXAMINE NEXT LOCATION	•	•	+	•	(LF)	•	(SP)	•
ALL MONITORS USE [illegible] MODE	CHANGE AND ADVANCE	(SP)$	(SP)$	(SP)$	[(SP)]$	/	(SP)	$	(SP)
	CHANGE DON'T ADVANCE	=$	=$	=$					
	BINARY DISPLAY	2	2	2					
*SEVERAL CHARACTERS MAY BE ENTERED	ENTER ASCII	"(ASC)	"(ASC)						
	EXIT MEMORY MODE	(CR)	(CR)	(CR)	(CR)	(CR)	(SP,CR)	(CR)	[illegible]
ALL OTHER COMMANDS MAY BE USED WHILE GMXBUG IS IN MEMORY MODE									
OTHER MEMORY FUNCTIONS	DUMP BLOCK FORMAT HEX	DS	DS	DS		DS			
	DUMP DISASSEMBLY FORMAT	FS	FS	FS					TS
	DUMP BLOCK FORMAT ASCII	DS	DS	DS					
	DISPLAY REGISTERS	R	R	R	R	R	R	R	R
	BLOCK MOVE	XS	XS						
	FILL MEMORY	ZS	ZS					IS	
* 1, 2 or 3 BYTES	BYTE SEARCH	HS*			FS**				
** 1 BYTE ONLY	TEST MEMORY	TS	TS	TS					
	INITIALIZE STACK POINTER	I①	IS	IS					
DEBUGGING	SET BREAKPOINT	BS#	BS#	BS#	BS	BS		KS	
	REMOVE BREAKPOINT	K#	K#	K#	B	B		K	
# BREAKPOINT NUMBER (0-3)	LIST BREAKPOINTS	P	P	P					
	CALCULATE CHECKSUM	CS	CS	CS					CS
	HEX ARITMETIC	AS	AS	AS					
EXECUTE USER PROGRAM	GOTO USER PROGRAM	GS	GS	GS	G	G	G	G	GS
	JUMP TO USER PROGRAM	JS	JS	JS	JS	ES		JS	
	JUMP TO USER PROM	U	U	U	Z	(ESC)		4	
TAPE I/O	LOAD FROM TAPE	L	L	L	[O] L	L	L	L	L[O]S
	LOAD W/OFFSET	LS	LS	LS					
	SAVE TO TAPE	SS	SS	SS	[O] P	PS	P	PS	P[O]S
DISK BOOT	DISK BOOT	②	O(SWT)		O(SWT)			O(SSB)	
I/O CONTROL	PRINTER OUTPUT	I①	*O(B,P,S)					H**	
	SET PRINTER NULLS	I①	NS						
* SERIAL PRINTER	OUTPUT WAIT	^S	^S	^S					^S
** USER MUST PROVIDE PRINT DRIVER ROUTINE	FULL/HALF DUPLEX	I①	H(H,F)					E or N	
	ERASE SCREEN	E	E	E	C				
UTILITY ROUTINES	EXTENDED UTILITIES	③	③	③		④		⑤	
	MIKBUG ENTRY POINTS	24	24	4	16	30	ALL	42	14
USER DEFINED SWI	USER DEF. SWI	YES	YES	YES	YES	NO	NO	NO	YES

LETTERS ARE USED TO CALL INDICATED FUNCTION. BRACKETS [ ] INDICATE OPTIONAL ENTRY
$ INDICATES HEX ADDRESS AND/OR DATA ENTRY.

(SP) = SPACE
(CR) = CARRIAGE RETURN
(LF) = LINE FEED
(ASC) = ASCII CHARACTERS
(ESC) = ESCAPE
^S = CONTROL S

SWTBUG is a trademark of Southwest Technical Products Corp.
RT/68 is a trademark of Microware Systems Corp.
MIKBUG is a trademark of Motorola Inc.
SMARTBUG is a trademark of Smoke Signal Broadcasting
MSIBUG is a trademark of Midwest Scientific Instruments
GMXBUG is a trademark of GIMIX INC

1 I(n) (COMMAND) A MULTI-FUNCTION COMMAND USED TO INITIALIZE PRINTER OUTPUT, NULLS, DUPLEX, OUTPUT WAIT, AND STACK POINTER OR REPORT THEIR STATUS
2 AVAILABLE ON SEPARATE PROM
3 IN ADDITION TO STANDARD UTILITIES GMXBUG INCLUDES AN EXTENDED UTILITY PACKAGE. ALL ARE ACCESSED THROUGH AN SWI CALL. TABLE DRIVEN LINKAGE PROVIDING INDEPENDENCE FROM ABSOLUTE ADDRESSING PLUS EASIER PRESERVATION OF MACHINE REGISTERS
4 RT/68 SUPPORTS REAL-TIME MULTI-TASKING
5 SMARTBUG INCLUDES REGISTER CHANGE AND A SINGLE STEP/TRACE FUNCTION

LOOKUP

A DATA MANAGER

MYCROFTWARE SYSTEMS HAS ANNOUNCED AN EASY TO USE DATA MANAGER FOR USERS OF THE FLEX OPERATING SYSTEM. LOOKUP WORKS WITH DATA RECORDS WITH THE ABILITY TO ADD, DELETE, INQUIRE, CREATE, PRINT, LIST, AND PURGE RECORDS. APPLICATIONS RANGE FROM A SIMPLE NAME AND ADDRESS LIST TO FINANCIAL OR INVENTORY RECORD APPLICATIONS.

DATA RECORDS MAY BE VARIABLE LENGTH WITH VARIABLE NUMBER OF FIELDS. DATA FILES MAY ALSO BE EDITED FOR MASSIVE OR COMPLEX CHANGES TO RECORDS. DATA IS ALSO ACCESSIBLE FROM BASIC FOR MATHEMATICAL MANIPULATIONS OR CUSTOM REPORTING.

LOOKUP COMES COMPLETE WITH 5 INCH MINIDISK, REFERENCE MANUAL, SAMPLE DATA FILE, SOURCE LISTING, AND SOURCE PROGRAM ON DISK. ALSO INCLUDED IS AN INDEX OF ALL 6800 ARTICLES FROM MAJOR HOBBY MAGAZINES FOR INQUIRY FROM DISK.

LOOKUP IS WRITTEN IN FAST ASSEMBLY LANGUAGE AND RUNS IN A MINIMUM MEMORY SYSTEM. LOOKUP IS AVAILABLE FOR $49.95 FROM MYCROFTWARE SYSTEMS, P.O. BOX 1138, ST. CHARLES MISSOURI, 63301.

COMMENTS ON SOME POPULAR FAIRY TALES

BY
Albert S. Jackson, Ph.D

Myth No. 1
The Z 80 is fast–WRONG. It actually takes 4 to 6 cycles just to fetch each op code compared to 1 cycle for the M6800. (1.0 to 1.5 µs for the Z 80A running at 4 MHz; 0.5 µs for the M68B00 running 2 MHz).

Myth No. 2
The Z-80 is a third generation microprocessor–WRONG–it is a rework of the 8080 with M6800 features added to improve performance. It is better than the 8080 or 8085, but definitely not a third generation microprocessor such as the M6809 (to be introduced in 1978).

Myth No. 3
The Z 80, because of its expanded instruction set, is easy to program–WRONG AGAIN–it has a very complex instruction set which is difficult to learn and use effectively.

Myth No. 4
Adequate development support hardware and software are available for the Z 80–WRONG. None of the Z 80 suppliers nor, for that matter, the independent suppliers of so called "Universal" development systems have invested the tremendous amount of capital required to produce adequate hardware and software support products for the Z 80. Intel has done a reasonable job in this area for the 8080 and 8085, but Motorola's support is truly outstanding with their broad line of EXORciser products, including by far the best disk operating system (MDOS), Fortran Compiler, MPL Compiler, Macro Relocatable Assembler, etc. . .

Myth No. 5
The Z 80 is applicable to a broad spectrum of product needs–WRONG, WRONG, WRONG–it fits only a narrow niche of applications, forcing its users to turn to completely different processors for either high volume cost sensitive products or for more complex high performance models of the same product. Only Motorola has solved the problem of matching the same hardware and software approach to a wide variety of needs, from the single chip MC6801, the two chip MC6802/MC6846, the multi chip MC6800/MC68A00/MC68B00 based systems, to the very high performance MC6809 based systems. The same development system, software and training are applicable to the entire M6800 family line. The Motorola approach offers the user tremendous savings in training costs, development systems costs, documentation costs, field support costs, etc.

M6800 vs Z 80

1. **Architecture**
The Z 80 adds an alternate set of eight 8 bit registers. (but it takes 4 cycles to exchange A F ↔ A' F' and four more for BCDEHL ↔ B' C' D' E' H' L'). The alternate registers are only marginally useful for interrupts because nested interrupts are not supported with this technique of register exchange. If maskable as well as the non-maskable interrupts are used, as is often the case, the alternate register set can be used with the one interrupt but not the other. The second source of interrupts must use, as a minimum, the PUSH A F and PUSH H L instructions for a total of 42 cycles (as opposed to 10 cycles for the exchange register method). All M6800 registers are automatically saved on interrupt, allowing unlimited nesting (10 cycles are used for these save and restore operations). The overhead for servicing even single, non nested interrupts is high for the Z 80– twice that of the M6800.

Two index registers have been added, perhaps the major improvement over the 8080. However, all indexed instructions are of the extended type, requiring 3 bytes instead of the 2 used with the M6800. The indexed instructions are also quite slow, requiring 19 or 20 cycles as opposed to the 5 to 7 cycles of the M6800. Loading an index register takes from 14 cycles (immediate) to 20 cycles (extended address). Comparable M6800 speeds are 3 and 5 cycles. Furthermore, the Z 80 has no compare index register instruction–an extremely damaging deficit.

An interrupt page address register (8 bit) has been added to allow interrupt routines to be located anywhere in memory–a feature the 6800 achieves through auto interrupt vectoring.

A memory reference register (7-bit) has been added to allow refresh during instruction execution cycles. This is marginally useful.

A second interrupt pin (non-maskable) has been added; this is the NMI of the M6800.

2. **Instruction Set**
The Z-80 uses the 12 unused 8080 op codes and expands with 2 byte op codes for a group of new instructions. However, many of the new instructions, such as block move, would seldom be used and have little effect on overall throughput. A group of double precision (2 byte) instructions have been added, but these are slow–the Z 80 is actually 68% slower than the M6800 in double precision add, for example.

The Z 80 does not have direct (base page) addressing, as does the M6800, but instead uses indirect register or "pointer" addressing with the H L register pair. In addition, the B C and D E register pairs can be used as pointers for accumulator-only load and stores. The net result is that more code is used with the Z 80 and the programming is more complicated.

The Z 80 offers bit set and reset instructions– which are actually slower than the use of the AND and OR immediate instructions with the M6800.

3. **Speed** (Beware of comparing MPUs by the clock frequency!)

Clock Rates are		
	2.5 MHZ	Z80
	4.0 MHZ	Z80A

4 to 6 cycles are used for op code fetches vs 1 cycle for the M6800. Thus the M6800 is 2 to 3 times faster at op code fetches even though the clock is ½ as fast.

Fastest instruction is 4 cycles for exchange A F ↔ A' F', etc., and load register immediate (compared to 2 cycles on M6800). Most instructions take longer than M6800. See table 1.

4. **Conclusions**
The Z-80 is an improvement over the 8080 or 8085 but because it was designed around the obsolete 8080 architecture in order to be software compatible with it, the Z-80 and Z-80A are inferior to the MC68A00 and MC68B00. In fact, even the MC6800 is pretty much a match for the Z-80A except for the seldom used block move type of application. The much touted Z-80A block I/O instruction is not really that much of an improvement over the MC68B00–180K vs 154K bytes/sec 256 bytes or less. In any case, use of the MC6844 DMA controller chip allows much faster I/O, block lengths up to 65K bytes and four simultaneous I/O operations.

The 16 bit arithmetic instructions of the Z-80 seem attractive at first, but the M6800 is actually faster in this type of operation (see table 1).

The Z-80A has about 5% more logic on a 6% larger die than the MC68B00. Third generation microprocessors, such as the MC6809 will have approximately 100% more logic on the die than present second generation microprocessors.

Use of the Z-80 to upgrade existing 8080 designs makes some sense. New design-ins of the Z-80 at this late stage do not make sense, however. Even if the user is "locked" into the 8080 because of prior use, he will have to soon retrain his people to use a different processor anyway because there is no way that this first generation architecture will be carried over to third generation microprocessors.

SPEED COMPARISON

TABLE I

	Z-80A %	6800		68B00	Z-80		Z-80A
			µs	µs		µs	µs
Load A direct (Z-80 extended)	−117	3	3.0	1.5	13	5.2	3.25
Load A, register indirect	− 17				7	2.8	1.75
Load A immediate	− 75	2	2.0	1.0	7	2.8	1.75
Load A indexed	− 90	5	5.0	2.5	19	7.6	4.75
Store A direct (Z-80 extended)	− 63	4	4.0	2.0	13	5.2	3.25
Store A, register-indirect	+ 13				7	2.8	1.75
Store A indexed	− 42	6	6.0	3.0	19	2.6	4.25
Add A to B	0	2	2.0	1.0	4	1.6	1.0
Add A immediate	− 75	2	2.0	1.0	7	2.8	1.75
Add direct to A, Z-80 register indirect (HL)	− 17	3	3.0	1.5	7	2.8	1.75
Add indexed	− 90	5	5.0	2.5	19	7.6	4.75
Jump extended	− 67	3	3.0	1.5	10	4.0	2.5
Jump to subroutine	+ 6	9	9.0	4.5	17	6.8	4.25
Return	0	5	5.0	2.5	10	4.0	2.5
Input to A (extd.)	− 38	4	4.0	2.0	11	4.4	2.75
Increment index reg.	− 25	4	4.0	2.0	10	4.0	2.5
Store index reg., direct	−100	5	5.0	2.5	20	8.0	5.0
Interrupt overhead start	−115	10	10.0	5.0	43	17.2	10.75
Interrupt overhead return (Includes EX AF and Store of one index reg.)	− 90	10	10.0	5.0	38	15.2	9.5
Add two 16-bit numbers stored in RAM and Store result	− 68	20	20.0	10.0	67	26.8	16.75

NOTE: $Z\text{-}80A\ \% = \dfrac{68B00\ \mu s - Z\text{-}80A\ \mu s}{68B00\ \mu s} \times 100\%$

EXAMPLE: Load a direct (Z-80 extended) $\dfrac{1.5 - 3.25}{1.5} \times 100\% = -117$

The Z-80A is 117% slower than the MC68B00 for this function.

COMPUTERWARE

PRODUCT DESCRIPTION		PRICE
RANDOM ACCESS DISK FILE BASIC w/EDIT, line input, & more new features	(SSB ONLY)	89.95
SUPER DISK FILE HANDLING BASIC w/EDIT, line input, & more new features	(FLEX ** ONLY)	49.95
SUPER CASSETTE FILE HANDLING BASIC	cassette	29.95
PROM RESIDENT CASSETTE FILE BASIC (SUPER features)	cassette 2716	100.00 250.00
PILOT - OUR VERY OWN FOR THE 6800 (with COMMENTED SOURCE LISTING) (also with SOURCE CODE ON DISK)	(SSB or FLEX)	24.95 37.95 49.95
DISK CHECK FILE MAINT. SYSTEM * (Complete checking system)	(SSB or FLEX)	49.95
DISK NAME AND ADDRESS SYSTEM * (Selective printing - labels)	(SSB or FLEX)	49.95
RANDOM MAILING SYSTEM V2 * (Commercial quality package) (Handles large data bases with ease)	(SSB ONLY)	89.95
DISK HOME INVENTORY SYSTEM * (Random access files - online inquiry) (6 Reports either hardcopy or display)	(SSB ONLY)	49.95
AMERICANA PLUS (14 NEWTECH Songs) (Dixie, Noel, Eyes of Texas, & more)	cassette (SSB or Flex)	15.95 19.95
CSS FOUR PART #1 (8 Songs - O'Holy Night) (Moonlight Sonata, Rhapsody in Blue, etc)	cassette (SSB or FLEX)	24.95 24.95
CSS FOUR PART #2 (7 Songs - Autumn Leaves) (Bumble Bee, Bach Preludes, Romeo/Juliet)	cassette (SSB or FLEX)	24.95 24.95
CSS FOUR PART #3 (7 Songs - McArthur Park) (The Sting, Windsong, This Guy's in Love)	(SSB or FLEX)	24.95
BASIC PGMS #1 (Bluff, Chase, Animal, Hamurabi) * (Mastermind, Biorythm, Horse race, etc.)	(SSB or FLEX)	19.95
BASIC PGMS #2 (Decision, Black Jack, Crash) * (Math Lesson, Lunar, Keno, Furs, Wumpus)	(SSB or FLEX)	19.95
BASIC PGMS #3 (Doall, Maze, Roadrace, Lifetime) * (Interest & Income Calc, Baseball, etc)	(SSB or FLEX)	19.95
WHIZ (TM) - High Speed Binary punch load	cassette	15.95

* source listing inc. - ** FLEX is a Trademark of TSC
--> COMPUTERWARE is a Registered Trademark of Computerware <--

****** Cassette format: AC-30 --- 8" disk add $2.00 ******
****** ALL SSB Software is available on 8" diskettes ******

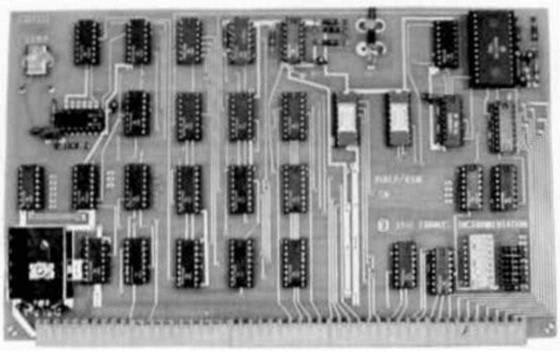
C O M P U T E R W A R E   S O F T W A R E   S E R V I C E S
830 First Street   Encinitas,   California   92024

PRODUCT DESCRIPTION			PRICE
LEARN ASSEMBLER - PART I (5 Lessons - Requires 16K)		cassette (SSB or FLEX)	19.95 19.95
LEARN BASIC PACKAGE (I II & III) (All 12 lessons - need 16K)		cassette (SSB or FLEX)	39.95 39.95
RENBAS - RENUMBER BASIC Programs Same features as below		cassette w/source list	24.95
RENBAS - RENUMBER BASIC Programs Selective starting number and increment value		(SSB or FLEX) w/source list w/source disk	 24.95 34.95
XREF - Assembly Program Label Cross-Reference - Complete Commented Source Listing		(SSB or FLEX) w/source list w/source disk	 24.95 34.95
SUBMIT - DOS Batch Processor Macro capabilities Dos Error Recovery		(SSB ONLY) w/source list w/source disk	 24.95 34.95
BFD-68 DISK TRANSIENTS #1 Includes: Listp, Init, Mem Lodhex, Savlod, Printp Cordmp, Disasm, Convrt, Epend		(SSB ONLY) w/source list w/source disk	 24.95 34.95
BFD-68 DISK TRANSIENTS #2 Includes: Print, Dir, Pip Cksm, Search, Fill, Filcom Delete - Wild Card Options!		(SSB ONLY) w/source list w/source disk	 24.95 34.95
BFD-68 DISK TRANSIENTS #3 Includes: Fdump, Sdump, Map, Scout, Memdump, Compare, Hardcopy for other transients		(SSB ONLY) w/source list w/source disk	 24.95 34.95
FLEX UTILITY COMMANDS #1 Includes: Examine, Findhexi, 2, 3 Dir, Kill, Files, Frags, Repeat		(FLEX ONLY) w/source disk	19.95 24.95
DOODLEBUG - EXTENDED SMARTBUG MONITOR * (with SMARTBUG in EPROM)		cassette 2716	19.95 69.95
SMARTBUG		(2708) (2716) 5v	39.95 59.95
EDITOR or ASSEMBLER EDITOR and ASSEMBLER	SE-1 OR SA-1 SE-1 & SA-1	cassette/disk cassette/disk	29.00 53.00
TEXT PROCESSOR		(SSB ONLY)	39.95
SSB BASIC AND RANDOM DOS		(SSB ONLY)	39.95
TRACE - DISASSEMBLER	TD-1	(SSB ONLY) cassette	25.90 19.95
SOURCE GENERATOR	SG-1	(SSB ONLY) cassette	29.90 24.95

COMPUTERWARE
830 FIRST STREET   ENCINITAS, CA 92024
the 6800 Specialist

GOOD DEAL EXTENDED!
PRICES SHOWN ARE FOR ORDERS RECEIVED BY JUNE 30

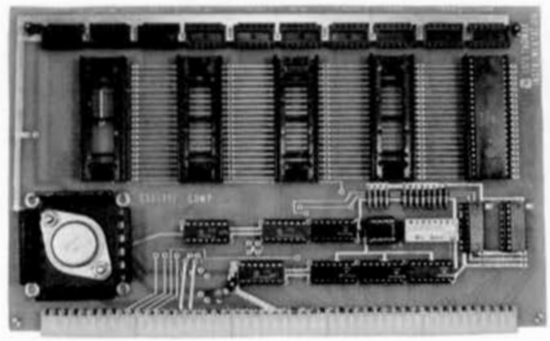
NORMAL PRICES APPROX. 10% HIGHER . . . CALL FOR DEALER, OEM, AND GROUP PRICING



SS-50 VIDEO RAM

- Fully Synchronous Operation to Eliminate Jitter
- Full 128 Set of 7 by 9 Characters & Reverse Image
- 1K of Memory can be Mapped in any 1K Increment
- 10 Pages of Documentation Including Software

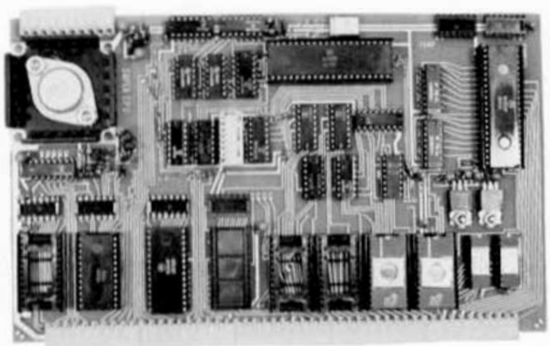
ASSEMBLED $135.00
CARD, CRYSTAL, & DOC $35.00



SS-50 PARALLEL I/O

- 2 Ports Expandable to 10 (Just Add PIA'S @ $6.95 ea.)
- Each Port has 8 I/O Plus 2 Control . . . 100 I/O Lines . . .
- Large Regulator Supplies Power to I/O Connectors
- Board Can be Mapped to any 32 Word Boundary

ASSEMBLED (W/1PIA) $89.00
CARD ONLY $32.00



SS-50 SUPER CPU

- SS-50 Buss . . or . . Stand Alone Computer
- 1K of Ram at $A000, I/O on Board at $A400 (Relocatable)
- 2K Monitor in 2708 Eprom (Expandable to 4K)
- 2 Parallel 8 bit ports with 2 Control Bits and Power
- 1 RS-232 ACIA Port (Expandable to 2nd TTL ACIA)
- 3-16 Bit Counter Timers (Expandable to 6 . . . Add 2nd 6840)
- Additional 128 Byte Ram at 0000 can be Jumpered out
- Battery Back-up Available on 32 Bytes of Ram
- Back/Back with other SS-50 Cards to Make Small Systems

ASSEMBLED (AS SHOWN) $179.00
CARD ONLY $44.00

Gimix 16K Without Soft Addressing $298.13
Gimix 16K Static Ram W/Softadd $368.16
Our SS-50 Wire-Rap Card $24.00

3, 4, and 7 slot SS-50 Backplanes
We also Make Non SS-50 Single Board 6800 Sys.

We have been in business for over 8 years, designing industrial machine tool controls and monitors . . . Call us for help with your product or laboratory instrumentation problem . . . we are dealers for:

GIMIX SWTPC SSB

***** LET US QUOTE YOU A PACKAGE PRICE *****

THOMAS INSTRUMENTATION

2709 Dune Drive, Avalon, N.J. 08202
Phone (609) 967-4280

VISA

Advanced Disk Operating and File Management System from PERCOM

INDEX™

(INterrupt Driven EXecutive)

only $99.95

A fast, adaptable disk operating program for 6800 computers featuring:

Fast execution — I/O devices are serviced by interrupt request. Eliminates polling time.

Adaptability — I/O devices and peripherals are accessed the same as disk files. New devices may be added without changing the operating system.

Versatility — An unlimited number of DOS commands may be added. Over 60 system entry points for program linkage provided.

System savings — INDEX™ and user-added commands and routines are disk-resident. No need to add memory for program enhancement.

Optional 108-page **Advanced Programmer's Guide**: I/O drivers & examples, describes system entry points, etc.... $45.00

Versions of INDEX™ are available for the PERCOM LFD-400™, SWTP MF-68 and Smoke Signal Broadcasting Company's BFD-68 disk systems, and for Motorola's EXORciser™ development system.

INDEX™ handles both ASCII and binary files, and disk files are automatically created, allocated and de-allocated.

Files are referenced by names — of up to eight characters — and file name parameters are addended for name extension (to further define the file), drive number, directory level and a file protection flag.

The INDEX™ operating system software also features a versatile BACKUP routine for copying files onto a diskette.

Console Interface Software

The console interface segment of INDEX™ software supports any standard serial ASCII terminal, and features:

- Program Interrupt (vs. Reset) for runaway programs.
- Operator Start, Stop and Skip display control.
- An interrupt-serviced, type-ahead character-queue buffer.
- A secondary line editing queue buffer.

INDEX™ Commands

Commands are given to INDEX™ a line at a time. Processing begins when the Return key is struck. The type-ahead character-queue buffer allows the operator to input a command while the previous command is being executed. A character may be deleted and corrected with the Backspace (Control H) key, and a line may be cancelled with the Rub-Out key. **In addition to the following commands, the user may add any number of his own commands.**

BACKUP · CONVERT · COPY · COUNT · DATE · DELETE · DIRectory · DISKEDIT · DISKINIT · EXAMINE · FILL · HELP · RENAME · SAVE · SETSTART · SETVERSION · SYSDISK · TYPE · USERDISK

System Requirements

System requirements are as follows:

1. 8K RAM at address $A000 — $BFFF
2. Minimum of 8K RAM beginning at address $0000
3. ACIA console interface (SWTP MP-C interface)
4. SWTBUG™ or equivalent monitor

INDEX is a trademark of PERCOM Data Company. LFD-400 is a trademark of PERCOM Data Company.

EXORciser is a trademark of the Motorola Corporation. SWTBUG is a trademark of Southwest Technical Products.

PERCOM

PERCOM DATA COMPANY, INC.
Dept. 68 211 N. Kirby Garland, TX 75042
(214) 272-3421

How to Get *Your* INDEX™ Software

INDEX™ is supplied on two mini-diskettes together with a Users Manual for $99.95, and may be ordered from PERCOM by dialing, toll-free, **1-800-527-1592.** In addition to checks and money orders, Visa and Master Charge credit cards are honored. Texas residents add 5% for sales tax.

PerCom *'peripherals for personal computing'*

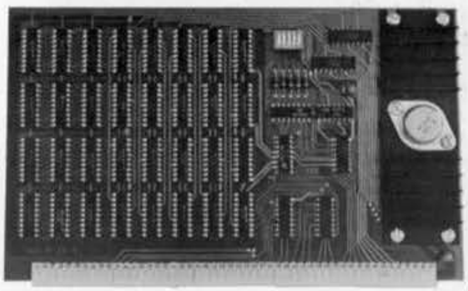
We're not just blowing smoke

SMOKE SIGNAL BROADCASTING PRESENTS IT'S

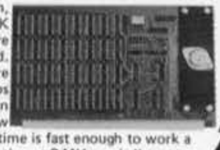
$299.00

M-16A STATIC MEMORY SYSTEM

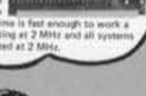
- Allows SWTPC 6800 expansion to 48K
- Low Power • Uses Single +8 Volt Supply
- SWTPC 6800 Plug Compatible • STATIC – No refresh required



The M-16A STATIC random access memory system, with a total storage capacity of 16834 words of 8 bits each, is switch selectable to any 4K starting address, and a hardware write protect switch is also included. The system's storage elements are 4K by 1 STATIC memory chips which store 4 times as much in only 12% more space than the low power 2102's. Typical access time is fast enough to work a 6800 based computer operating at 2 MHz and all systems are factory tested at 2 MHz.



The M-16A STATIC random access memory system, with a total storage capacity of 16834 words of 8 bits each, is switch selectable to any 4K starting address, and a hardware write protect switch is also included. The system's storage elements are 4K by 1 STATIC memory chips which store 4 times as much in only 12% more space than the low power 2102's. Typical access time is fast enough to work a 6800 based computer operating at 2 MHz and all systems are factory tested at 2 MHz.



The M-16A STATIC random access memory system, with a total storage capacity of 16834 words of 8 bits each, is switch selectable to any 4K starting address, and a hardware write protect switch is also included. The system's storage elements are 4K by 1 STATIC memory chips which store 4 times as much in only 12% more space than the low power 2102's. Typical access time is fast enough to work a 6800 based computer operating at 2 MHz and all systems are factory tested at 2 MHz.

SMOKE SIGNAL BROADCASTING®

31336 Via Colinas, Westlake Village, CA 91361, (213) 889-9340

SMOKE SIGNAL BROADCASTING

31336 Via Colinas, Westlake Village, CA 91361
(213) 889-9340

☐ Send information on your M-16A
☐ Send name of nearest dealer

Name ____________________
Address ____________________
Company ____________________
City ____________________
State/Zip ____________________

'68' Micro Journal
3018 Hamill Rd.
Hixson, TN 37343

Second Class Postage
Paid at Chattanooga TN